E. Floegel

FORTH®

Handbuch

Grundlagen

Einführung

Beispiele

OK

```
SCR# 2
( ADRESSEN EINGABE  FORTS.   EF )
64 CONSTANT RECLEN

: EIN PAD + SWAP  EXPECT ;
: (EINGABE) HOME CR PAD RECLEN 32 FILL
   "  VORNAME " VORNAME EIN    CR
```

```
0 VARIABLE WO  1 VARIABLE #NR
: ANFANG 1 #NR ! ;
: ['] [COMPILE] ' ;
: (VERGL) ( ADR1 ADR2 C --> F )
    BEGIN ROT DUP C@ >R OVER I = R> SWAP
          DUP IF 0
          ELSE DROP >R ROT DUP C@ R>  =
                  WHILE 2DROP 1+
                  DROP R> ;
: LEER PAD RECLEN 32 FILL ;
: WAS ['] 2 - EXECUTE SWAP DROP DUP WO !
  1 WO ... PAD + SWAP
  C@ ...

 OK
```

ISBN 3–911682–88–6

Es kann keine Gewähr dafür übernommen werden, daß die in diesem Buche verwendeten Angaben, Schaltungen, Warenbezeichnungen und Warenzeichen, sowie Programmlistings frei von Schutzrechten Dritter sind. Alle Angaben werden nur für Amateurzwecke mitgeteilt. Alle Daten und Vergleichsangaben sind als unverbindliche Hinweise zu verstehen. Sie geben auch keinen Aufschluß über eventuelle Verfügbarkeit oder Liefermöglichkeit. In jedem Falle sind die Unterlagen der Hersteller zur Information heranzuziehen.

Nachdruck und öffentliche Wiedergabe, besonders die Übersetzung in andere Sprachen verboten. Programmlistings dürfen weiterhin nicht in irgendeiner Form vervielfältigt oder verbreitet werden. Alle Programmlistings sind Copyright der Fa. Ing. W. Hofacker GmbH. Verboten ist weiterhin die öffentliche Vorführung und Benutzung dieser Programme in Seminaren und Ausstellungen. Irrtum, sowie alle Rechte vorbehalten.

COPYRIGHT by Ing. W. HOFACKER © 1982,
Tegernseerstr. 18, 8150 Holzkirchen/Obb.

1. Auflage 1982
Gedruckt in der Bundesrepublik Deutschland – Printed in West-Germany – Imprime'en RFA.

VORWORT

FORTH ist eine neue Computersprache. Sie entspricht mehr der gesprochenen Sprache als irgendeine andere höhere Programmiersprache. Dennoch kann auf jedes Bit im Rechner zugegriffen werden.

FORTH läßt dem Programmierer viel Freiheit. Das ist ein großer Vorteil, denn er kann sich seine eigenen Datenstrukturen schaffen, er kann den Compiler verändern und sogar den Compile-Vorgang beeinflussen. Dies ist aber auch ein Nachteil, den nun ist der Programmierer voll für den Programmablauf verantwortlich.

FORTH weist einige einzigartige Besonderheiten auf:

- Es ist kompakt.
- Es ist sein eigenes Betriebssystem, es ist Interpreter und Compiler.
- Es ist schnell und es ist strukturiert (kein Sprungbefehl).
- Es ist transportabel.

Die Programmentwicklungszeit wird erheblich gegenüber anderen Programmiersprachen verkürzt, und es ist transportabel. Ein für einen bestimmten Rechner geschriebenes Programm läßt sich ohne Schwierigkeiten auf einem anderen Rechner umsetzen.

FORTH hat viele Anwendungsmöglichkeiten. Von der Steuerung von Teleskopen über die Steuerung von Orangenpflückmaschinen, Trickfilmkameras bis hin zu Datenbanken sind einige Anwendungsmöglichkeiten, bei den FORTH anderen Programmiersprachen weit überlegen ist.

FORTH ist keine leichte Programmiersprache. Schwierigkeiten ergeben sich hier, wenn man Vergleiche mit anderen Sprachen sucht. FORTH ist eingenständig und unterscheidet sich durch sein Wörterbuchkonzept grundsätzlich von anderen Sprachen.

Danken möchte ich der Firma FORTH INC. die es mir ermöglichte, an einem ihrer Seminare teilzunehmen und Frl. Grünwald. Sie hatte beim Setzen hart mit den Spracheingenheiten von FORTH zu kämpfen.

Dem Leser wünsche ich viel Spaß beim Lesen, am besten in der FORTH-Schreibweise selbst

LESEN FORTH IF SPASS ELSE NOT THEN ;

Holzkirchen, Herbst 1982 E. Flögel

Wo gibts FORTH ?

Für die Beispiele im Buch können Sie folgende FORTH-Versionen verwenden:

1. FORTH 79
   für Apple II auf Diskette
   Micromotion Los Angeles, CA 90025, (231) 821–4340

2. FIG FORTH
   FORTH Interest Group
   P.O. Box 1105, San Carlos CA 94070
   Listings für 8080, 6809, 6800, 6502, 68000
   Fertige Versionen auf Disk für TRS-80/Video Genie und ATARI 800 über Ing. W. Hofacker zu beziehen.

3. QS-FORTH
   Speziell für ATARI-800 mit Grafik- und Soundeigenschaften
   Hersteller: Quality Software, 6660 Reseda Blvd, Suite 105, Reseda, CA 91335
   In Deutschland bei einigen ATARI-Fachhändlern verfügbar.

4. APEX-FORTH von ATARI direkt und dessen Fachhandel

5. Poly-FORTH von FORTH, Inc.
   2309 Pacific Coast Hay., Hermosa Beach, CA 90254, (213) 372–8493
   Entwicklungsversionen für 16-Bit Personalcomputer in Vorbereitung.

6. LERN-FORTH
   Erweiterte FIG-FORTH Version implementiert von E. Flögel/Hofacker Verlag für Apple II auf Diskette u. Cassette, VC-20 Commodore auf Cassette (keine virtuelle Speicherverwaltung) vom Ing. W. Hofacker Verlag, Holzkirchen.

7. ELCOMP-FORTH
   Erweiterte Fig-FORTH-Version mit virtueller Speicherverwaltung und Editor. für Apple II Disk, für ATARI 800 Disk, vom Hofacker Verlag, Holzkirchen.

8. PHS-FORTH 4.3
   (enthält u. a. FIG FORTH, Macro Assembler, Editor, Reals, Floppyhandling, etc.)
   PHS/SLS, Davenstedter Str. 8, 3000 Hannover 91, Tel. 0511/453817

9. FORTH für OSBORNE-I
   von der Firma Osborne in München, Tel.: 089/4705068

10. 8080/780/CPM-FORTH
    Laboratory Microsystems, 4147 Beethoven St., Los Angeles, CA 90066, (213) 390–9292

11. Superforth von Hayden
    ohne Editor, jedoch Floating Point und Apple Grafik erhältlich vom Hofacker Verlag.

Inhaltsverzeichnis

1. Was ist FORTH ? 1

2. Sprachelemente von FORTH 5
   2.1 Allgemein verwendete Bezeichnungen 5
   2.2 Worte in FORTH 6
   2.3 Die postfix oder umgekehrte polnische Notation 7
   2.4 Arithmetische Befehle 11
   2.5 Befehle zur Veränderung des Stapels 13
   2.6 Definition eigener Worte 16
   2.7 Konstante und Variable 23
   2.8 Übungen und Beispiel 28
   2.9 Zusammenstellung der bisherigen Definitionen 34

3. Programmschleifen und Verzweigungen 41
   3.1 DO . . . LOOP 41
   3.2 Vergleichsbefehle 45
   3.3 IF . . . ELSE . . . ENDIF 46
   3.4 BEGIN . . . UNTIL 47
   3.5 BEGIN . . . WHILE . . . REPEAT 48
   3.6 Programmbeispiele 51
   3.6.1 Berechnung des GGT zweier Zahlen 51
   3.6.2 Fibboconacci-Zahlen 52
   3.6.3 Berechnung der Primzahlen zwischen zwei Grenzen 53
   3.6.4 Hexdump 56
   3.7 Zusammenstellung der Definitionen 59

4. Zahlensysteme 63
   4.1 Skalierung von Rechenabläufen 63
   4.2 Doppelt lange Zahlen 66
   4.3 Gemischte Zahlenoperatoren 68
   4.4 Zahlen ohne Vorzeichen 69
   4.5 Formatierung von Zahlenausgaben 71
   4.6 Gleitkommazahlen 74
   4.7 Zusammenstellung der Definitionen 75

5. Textfeld (SCREEN) und Editor **79**
5.1 Textfeld (Screen) 79
5.2 Der Editor 83
5.3 Zusammenstellung der Definitionen 86

6. Das Wörterbuch **89**
6.1 Speicherbelegung 89
6.2 Ändern des TOD 91
6.3 Speicherplatzreservierung 95
6.4 Virtueller Speicher 101
6.5 Aufstellen eines eigenen Wörterbuchs 104
6.6 Definitionsworte 106
6.7 Ändern des Compile-Verhaltens 111
6.8 Zusammenstellung der Definitionen 114

7. Texteingabe **117**
7.1 Zusammenstellung der Definitionen 121

8. FORTH Beispiele auf Heim Computern **123**
8.1 SOUND mit dem ATARI 800 124
8.2 FORTH auf APPLE II 133
8.3 FORTH auf AIM-65 140

9. Programmbeispiele **143**
9.1 Hilfsprogramme 143
9.2 Adressenkartei 145
9.3 Datenbank 151
9.4 Lagerverwaltung 155

ANHANG **163**
A Zusammenstellung der Definitionen 163
B Vorschläge zur Namensgebung 175
C Vorschläge zum Programmeintrag in ein Textfeld 177
D Spiel-Programm in PHS-FORTH für CBM-Computer 180
E FORTH-Referenzkarte 186

1 Was ist FORTH?

1. Was ist FORTH ?

FORTH ist wie BASIC, FORTRAN oder PASCAL eine Computersprache. Allerdings kann man sie nicht mit diesen Sprachen vergleichen. Diese Sprache wurde von CHARLES H. MOORE vor etwa 10 Jahren erfunden und von ihm zur Steuerung des Teleskops am Kitt Peak Observatorium verwendet. Die Lösung von Echtzeitproblemen, Maschinensteuerungen, Steuerungsaufgaben sind die wichtigsten Einsatzgebiete von FORTH. Der Programmablauf ist nicht ganz so schnell wie in Assemblersprache aber wesentlich schneller als in jeder anderen höheren Programmiersprache.

FORTH ist ein vollständig selbständiges Sprachsystem, das sowohl Compiler als auch Interpreter ist und sein eigenes Betriebssystem beinhaltet. Was ist nun das einzigartige an FORTH ?

Der Grundbegriff der hinter dieser Sprache steht, ist der Begriff des Stapels. Was ein Stapel ist, dürfte jedem bekannt sein, denn auf jedem anständigen Schreibtisch ist ein solcher Stapel. Auf diesem wird oben etwas abgelegt und vielleicht später wieder heruntergenommen. Es ist das Prinzip des LIFO-Speichers, Last In First Out. FORTH benutzt solche Stapel in zweierlei Weise. Es gibt den Stapel für Zahlen, auf dem auch die Berechnungen ausgeführt werden und den Stapel der definierten Wörter, dem Wörterbuch. Die Verwendung eines Stapels für Zahlen wäre an und für sich nichts Neues. Jeder HP-Taschenrechner verwendet einen Stapel zum Speichern von Zahlen. Das andersartige an FORTH ist das Wörterbuch.

Die Sprache besteht aus Worten, die nach ihrer Definition, das heißt, nach der Festlegung dessen, was dieses Wort bedeuten soll, auf dem

Definitionsstapel, dem Wörterbuch abgelegt werden. Dabei können bei der Definition des Wortes schon vorher vereinbarte Worte verwendet werden. Der Start eines Programms besteht dann im Aufruf eines Wortes. Dieses einfache Wörterbuchkonzept hat dazu geführt, daß FORTH auch bei Sprachübersetzern verwendet wird. Dies sind kleine Taschenrechner, bei denen nach der Eingabe eines Wortes, dieser Begriff in einer anderen Sprache auf der Anzeige ausgegeben wird. Wir werden später sehen, wie einfach sich dies in FORTH programmieren lässt.

Diese Möglichkeit, sich seine eigenen Sprachelemente selbst zu definieren, führt aber dazu, daß die Sprache laufend erweitert wird. Jedes FORTH-Programm beinhaltet also immer Worte, die in dem Programmpaket enthalten sind, das man gekauft hat und eigene Worte, die man hinzugefügt hat. Man kann aber auch Worte wieder aus dem Wörterbuch entfernen. Das Wort zum Löschen von Worten heißt FORGET.

Der Aufruf von FORGET ALLES löscht alle Worte, beginnend bei ALLES bis zum obersten Eintrag in den Stapel. Es wird also ein Teil des Stapels abgetragen. Das Löschen eines einzigen Wortes im Stapel ist nicht möglich und auch nicht sinnvoll. Denn dieses Wort kann Bestandteil eines oder mehrer Wörter, die im Stapel darüber liegen, sein. Auf die Möglichkeit, daß Worte mehrfach in einem Wörterbuch auftreten können und welche Auswirkungen dies auf den Programmablauf hat, werden wir später noch eingehen.

Bei den Berechnungen werden die Zahlen ebenfalls über einen Stapel verarbeitet. Dabei wird für die Zahlen und die Rechenzeicheneingabe die postfix oder auch UPN, die polnische umgekehrte Notation verwendet. Diese Programmierweise dürfte wiederum allen HP-Taschenrechnerbesitzern bestens bekannt sein. Wenn zwei Zahlen addiert werden, so wird z. B. in BASIC die infix Notation 2 + 4 verwendet, während FORTH die postfix Notation 2 4 + anwendet. In dem nächsten Kapitel wird darauf noch eingegangen werden.

In diesem Buch werden wir weitgehenst die poly-FORTH Version von FORTH INC. verwenden. Andere FORTH-Versionen sind fig-FORTH und FORTH 79. fig-FORTH ist eine von der FORTH INTEREST GROUP implementierte FORTH Version, die für die meisten Prozessortypen als Listing bezogen werden kann. Die FORTH

INTEREST GROUP bemüht sich, eine Standardisierung von FORTH zu erreichen. Sie gibt eine, zweimonatlich erscheinende Zeitschrift FORTH DIMENSIONS heraus. FORTH 79 ist eine Erweiterung von fig-FORTH und wurde von der FORTH STANDARDS GROUP erarbeitet. Es wäre nun falsch, hier zu glauben, daß es verschiedene Versionen von FORTH gibt. Der Sprachumfang kann verschieden sein, das ist aber in FORTH keine Einschränkung. Wie wir bald sehen werden, lassen sich Unterschiede durch die Definition von fehlenden Worten leicht beseitigen. Die hier im folgenden angeführten Programme und Beispiele sind hauptsächlich auf ATARI 800 und APPLE II erstellt worden. Für den ATARI 800 wurde ein fig-FORTH und für den APPLE II ein FORTH 79 verwendet. Auf Unterschiede zu poly-FORTH wird im Einzelfall hingewiesen.

FORTH-Programme sind nicht einfach zu beschreiben. Da die Wahl der Worte unbeschränkt ist, und da meist die Schreibarbeit eingeschränkt wird, führt dies oft zu einer Programmstenografie. Das Programm ist nur dem Programmierer verständlich und diesem, ohne die notwendige Dokumentation, auch nur kurze Zeit. Nach wenigen Tagen kann man oft sein eigenes Programm nicht mehr lesen. Dies ist bestimmt ein Nachteil von FORTH, der aber durch viele Vorteile gegenüber anderen Sprachen wieder aufgehoben wird.

Zum Schluß soll noch die Herkunft des Wortes FORTH geklärt werden. FORTH ist die 5-Buchstabenabkürzung von fourth in der Bezeichnung fourth generation of computer. Die Rechner, die Ende der 60er Jahre installiert wurden, trugen damals die Bezeichnung "Rechner der 4. Generation" (Erst Relais, dann Röhren, Tansistoren, IC). Da aber die 1130, auf der FORTH zum erstenmal implementiert wurde nur Bezeichner von 5 Buchstaben zuließ, wurde aus fourth FORTH (nach FORTH DIMENSIONS, VOL 1, NR 6).

2 Sprachelemente von FORTH

2. Sprachelemente von FORTH

In diesem Kapitel soll auf die Sprachelemente, Worte von FORTH eingegangen werden, wie sie in fig-FORTH oder FORTH-79 verwendet werden. Das Wort STAPEL bezieht sich ab jetzt auf den Stapel für Zahlen, dem Stapel der definierten Worte wird mit Wörterbuch bezeichnet.

2.1. Allgemein verwendete Bezeichnungen

Bei der Beschreibung des Wörterbuches und der Operationen mit dem Stapelspeicher werden Bezeichnungen verwendet, wie sie im Installation Manual für FORTH festgelegt sind.

Diese Bezeichnungen sind:

adr	Speicheradresse
b	8 bit Byte (die oberen 8 Bit sind Null)
c	7 bit ASCII-Zeichen
d	32 bit doppelt lange Festkommazahl mit Vorzeichen
f	Bool'sche Variable 0 = "false"
n	16 Bit Zahl mit Vorzeichen
u	16 Bit Zahl ohne Vorzeichen

Bei der Beschreibung des Wortes wird folgendes Schema angewendet:

WORT (STAPEL VORHER --) STAPEL NACHER)

KOMMENTAR

Beispiel:

. (n1 n2 n3 -- > n1 n2)

n3 wird auf dem Bildschirm ausgegeben.

Der . ist in FORTH die Anweisung zur Ausgabe des obersten Zeichens im Stapel auf den Bildschirm. Dies ist also die PRINT-Anweisung. Vor der Ausführung des Befehls befinden sich 3 Zahlen n1, n2, n3 im Stapelspeicher. Davon ist die am weitesten rechts stehende Zahl im TOS (Top of Stack). In unserem Fall also n3. Nach der Ausführung des Befehls sind nur noch zwei Zahlen (n2, n2) im Stapel, mit n2 an oberster Stelle. In den meisten Fällen wird noch ein Kommentar zu diesen Wortdefinitionen hinzugefügt werden.

2.2. Worte in FORTH

Als Wort in FORTH kann jede Buchstaben-, Zeichenkombination die zwischen Leerzeichen eingeschlossen ist, verwendet werden. Die Leerzeichen spielen in FORTH eine wichtige Rolle als Berenzer (delimiter) für Worte.

Gültige Worte sind also:

ALLES, 12IG, #?#, -- >, !, «, USW,

Nicht gültig ist dagegen:

1 ABC

es sei denn, das Leerzeichen nach der 1 ist ein, auf dem Bildschirm, nicht darstellbares Kontrollzeichen.

Die Einführung solcher Kontrollzeichen ist zwar möglich aber nicht sinnvoll. Wenn man versucht, ein fremdes FORTH-Programm zu lesen, so wird man sich vor allen nach den auftretenden Leerzeichen orientieren. Dabei verwirrt dann so eine Wortkonstruktion mit einem nicht sichtbaren Kontrollzeichen ungemein. Im Prinzip ist es möglich, ein Wort zu definieren, wobei auf dem Bildschirm außer dem Beginn und dem Ende der Definition nichts zu sehen ist.

Diese freie Wahl der Zeichen ist natürlich aber auch sehr tückisch, denn durch einen übershenen Tippfehler kann ein völlig neues Wort entstehen.

FORTH ist eine sehr nachsichtige Sprache. Alles, was eingegeben und nicht direkt als falsch erkannt wird, wird angenommen und eventuell sogar ins Wörterbuch eingetragen. Man darf also nicht blindlings darauf los programmieren, sondern man muß sich einigermaßen überlegen, was und wie man es programmiert.

2.3. Die postfix oder umgekehrte polnische Notation

Bei den meisten Computern erfolgt die Eingabe einer arithmetischen Berechnung in der gewohnten Form, wie z. B. bei der Addition zweier Zahlen:

4 + 7

Bei der postfix oder auch UPN Notation wird der Operator nach den Zahlen eingegeben:

4 7 +

Betrachten wir auch folgende Berechnung:

(3 + 5) * (4 – 2) – 7

Der BASIC-Interpreter, der eine solche Eingabe entschlüsseln muß, fängt links an und stellt fest, daß der Ausdruck mit dem Zeichen (beginnt. Als nächstes liest er die 3 und dann das + Zeichen. Diese Rechenoperation kann er nicht sofort ausführen sondern er muß erst die nächste Zahl, die 5, einlesen. Nun addiert er diese beiden Zahlen und liest als nächstes Zeichen das). Die Berechnung des ersten Terms ist abgeschlossen und das Ergebnis wird zwischengespeichert. In dieser Weise wird der restliche Teil der Ausdruck abgearbeitet. Der Rechner muß sich merken, daß eine Multiplikation mit dem nächsten Term durchgeführt werden muß. Diesen muß er zuerst entschlüsseln und berechnen. Dann kann er die Multiplikation ausführen und zum Schluß 7 abziehen. Ein Programm, daß eine Entschlüsselung eines Ausdrucks

dieser Art durchführen muß, ist relativ umfangreich. Wird dagegen der Ausdruck in der UPN-Notation eingegeben, so ist das Programm zur Berechnung sehr einfach.

Dem Ausdruck

(3 + 5) * (4 – 2) – 7

entspricht die Eingabe in UPN

3 5 + 4 2 – * 7 –

In Abbildung 2.1 ist gezeigt, wie diese Berechnung im Stapel abläuft.

STAPEL					TOS	EINGABE
					3	3
				3	5	5
					8	+
				8	4	4
			8	4	2	2
				8	2	–
					16	*
				16	7	7
					9	–

Abbildung 2.1
Berechnung eines arithmetischen Ausdrucks mit UPN

Durch den Operator werden also jeweils die beiden obersten Zahlen im Stapel verknüpft. Dadurch kann die Angabe von Klammern entfallen.

Aus (4 * 2 + (5 – 2)) / (6 – 1)

wird 4 2 * 5 2 – + 6 1 – /

In Abbildung 2.1 wird auch die Funktion des Stapels gezeigt. Jede

Zahl wird bei der Eingabe in den TOS (Top of Stack) übernommen, alle schon gespeicherten Zahlen werden dabei eine Stelle tiefer in den Stapel geschoben. Wird ein Rechenzeichen erkannt, so werden die beiden obersten Zahlen verknüpft und das Ergebnis wird im TOS gespeichert. Alle anderen Zahlen rücken dabei eine Stelle höher. Diese Art der Speicherung bezeichnet man auch als LIFO, Last In First Out. Die Zahl, die als letzte eingegeben wurde, ist die Erste bei einer Ausgabe.

Wir haben schon gesehen, daß ein . die Ausgabe der Zahl im TOS auf den Bildschirm bewirkt.

Die Anweisung PRINT 5 in BASIC lautet in FORTH 5 .

Wenn wir also

1 2 3 . . . CR (CR = RETURN-TASTE)

eingeben, so wird der Computer uns

3 2 1 OK

ausdrucken. Das OK bedeutet bei den meisten FORTH-Versionen, daß die eingegebene Befehlsfolge erfolgreich abgeschlossen wurde.

Weitere einfache Ausgabeanweisungen sind:

.“ XXX “ (n --> n)

Ausgabe des Textes XXX

Nach dem Wort .“ muß unbedingt ein Leerzeichen folgen. Der Text wird mit “ abgeschlossen.

Beispiel:

5 4 + .” 5+4= “. 5+4= 9 OK

Das Wort SPACE gibt ein Leerzeichen aus. Sollen mehrere Leerzeichen ausgegeben werden, so geschieht dies durch das Wort SPACES. Die Zahl der Leerzeichen muß vorher auf dem Stapel geschrieben werden.

SPACES (n1 n2 n3 -- > n1 n2)

n3 Leerzeichen werden ausgegeben.

Das Wort CR erzeugt eine neue Zeile.

Durch das Wort .R kann eine Zahlenfolge rechtsbündig in einem n Zeichen breiten Zahlenfeld ausgegeben werden. Die Breite n muß vorher auf dem Stapel sein.

Beispiel:

```
1 100 1000 CR . CR . CR .
```

ergibt

```
1000

100

1  OK
```

Dagegen

```
1 100 1000 CR 5 .R CR 5 .R CR 5 .R
```

ergibt

```
1000

 100

   1 OK
```

. R (n1 n2 n3 -- > n1 n2)

Tabulator wird auf n3 Stellen gesetzt. Die Zahl wird rechtsbündig ausgegeben.

Eine weitere Ausgabefunktion ist das Wort EMIT. Dieses Wort interpre-

nimmt die oberste Zahl auf dem Stapel als 7 bit ASCII-Zeichen und gibt sie auf den Bildschirm aus.

Beispiel:

```
64 EMIT @ OK
70 EMIT  F OK
192 EMIT  @ OK
```

EMIT (n1 n2 c --> n1 n2)

c wird als 7 bit ASCII-Zeichen ausgegeben.

2.4. Arithmetische Befehle

Für die 4 Grundrechnungsarten werden die Zeichen +, –, *, / verwendet.

+ (n1 n2 n3 --> n1 n)

n = n2 + n3

-- (n1 n2 n3 --> n1 n)

n = n2 -- n3

* (n1 n2 n3 --> n1 n)

n = n2 * n3

/ (n1 n2 n3 --> n1 n)

n = n2 / n3

Beispiel:

```
4 2 * 5 3 - + 6 1 / . CR   2 OK
```

In diesem und den folgenden Beispielen wird das, was vom Computer

ausgegeben wird, unterstrichen. Bei der Division ist zu beachten, daß diese eine Ganzzahldivision und keine Gleitkommadivision ist. Der Zahlenbereich reicht von –32768 bis 32767 .

Wird bei einer Division der Rest dieser Division benötigt, so erhält man ihn durch das Wort MOD.

MOD (n1 n2 n3 --> n1 n)

n ist der verbleibende Rest von n2 / n3

Benötigt man Quotient und Rest einer Division, so lautet das Wort hierfür / MOD.

/ MOD (n1 n2 n3 --> n1 n4 n)

n = n2 / n3 n4 = Rest.

Beispiel:

```
2 3 / MOD . . CR   0 2 OK
```

MAX (n1 n2 n3 --> n1 n)

n ist die größere der beiden Zahlen n2 und n3

MIN (n1 n2 n3 --> n1 n)

n ist die kleinere der beiden Zahlen n 2 und n3

Beispiel:

```
5 6 7 MIN . 6   OK
5 6 7 MAX . 7   OK
```

ABS (n1 n2 n3 --> n1 n2 n)

n = Betrag von n3

MINUS (n1 n2 n3 -- > n1 n2 –n3)

n3 ist negativ

Logische Befehle

Die logischen Befehle werden bitweise ausgeführt.

AND (n1 n2 n3 -- > n1 n)

n = n2 & n3

OR (n1 n2 n3 -- > n1 n)

n = n2 V n3

XOR (n1 n2 n3 -- > n1 n)

n = n2 Λ n3

Beispiel:

```
4 3 AND . 0  OK
4 3 OR . 7  OK
4 5 XOR . 1  OK
```

2.5. Befehle zur Veränderung des Stapels

Durch die Reihenfolge bei der Eingabe und durch die Ausführung von Operationen ist die Zahlenfolge im Stapel festgelegt. Für eine weitere Berechnung benötigt man oft eine andere Zahlenfolge. Wenn man zum Beispiel durch den . Befehl die oberste Zahl auf den Bildschirm ausgibt, so ist sie vom Stapel verschwunden. Benötigt man diese Zahl aber für die Weiterrechnung, so muß man sie duplizieren. Dafür gibt es den Befehl DUP.

DUP (n1 n2 --> n1 n2 n2)

DUP dupliziert das oberste Element des Stapels.

Beispiel:

```
2 4 DUP . . .  4 4 2  OK
```

Bei einer Subtraktion wird immer die oberste Zahl im Stapel von der nächsten Zahl abgezogen. Soll aber diese Subtraktion gerade anders herum ausgeführt werden, so muß man die Reihenfolge der beiden Zahlen vertauschen. Der Befehl dazu lautet SWAP.

SWAP (n1 n2 n3 --> n1 n3 n2)

SWAP vertauscht die beiden obersten Elemente.

Beispiel:

```
1 2 3 SWAP . . .  2 3 1  OK
```

Der Befehl OVER kopiert die 2. Zahl im Stapel in den TOS

OVER (n1 n2 n3 --> n1 n2 n3 n2)

OVER kopiert das zweite Element in den TOS.

Beispiel:

```
1 2 3 OVER . . . .  2 3 2 1  OK
```

Die Reihenfolge der obersten 3 Zahlen im Stapel kann durch den Befehl ROT verändert werden.

ROT (n1 n2 n3 --> n2 n3 n1)

ROT vertauscht zyklisch im Gegenuhrzeigersinn die obersten drei Elemente.

Beispiel:

```
1 2 3 ROT . . .  1 3 2  OK
```

Mit ROT wird keine der Zahlen dupliziert, sondern sie werden nur zyklisch vertauscht.

Soll die oberste Zahl vom Stapel entfernt werden, ohne sie auf den Bildschirm auszugeben, so geschieht dies mit dem Befehl DROP.

DROP (n1 n2 n3 --> n1 n2)

DROP löscht das oberste Element.

In FORTH-79 gibt es noch zwei weitere Befehle zur Veränderung des Stapels, die in fig-FORTH nicht definiert sind. Mit n PICK wird das n-te Element in die oberste Stelle kopiert.

n PICK (n1 n2 n3 --> n1 n2 n3 n4)

n4 = n-tes Element des Stapels.

Beispiel:

```
1 2 3 4 3 PICK . 2  OK
```

1 PICK entspricht DUP und 2 PICK entspricht OVER.

Der Befehl n ROLL holt das n-te Element an die Spitze und verschiebt die vorangehenden Elemente um eine Stelle nach unten.

n ROLL (n1 n2 n3 n4 --> n2 n3 n4 n1)

mit n = 4

2 ROLL entspricht SWAP und 3 ROLL entspricht ROT.

Für den internen Programmablauf verwendet FORTH noch einen weiteren Stapel, den sog. RETURN-Stapel. Dort werden Adressen,

Laufparameter und Rechenergebnisse zwischengespeichert. Dieser Stapel kann, wenn auch mit Vorsicht, zur Zwischenspeicherung von Zahlen verwendet werden.
Dazu stehen 3 Worte zur Verfügung:

〉R (n1 n2 n3 -- 〉 n1 n2)

n3 wird auf den RETURN-Stapel geschrieben.

R 〉 (n1 n2 -- 〉 n1 n2 n3)

n3 wird aus dem RETURN-Stapel zurückgeholt.

R (n1 n2 -- 〉 n1 n2 n3)

Das oberste Element des RETURN-Stapels wird auf den Rechen-Stapel kopiert.

Beispiel:

```
1 2 3 4 >R SWAP R> . . . .  4 2 3 1  OK
```

Die oberste Zahl des Stapels wurde vor dem SWAP im RETURN-Stapel gespeichert und hinterher wieder zurückgeholt. Durch SWAP wurde also nur 2 und 3 vertauscht. Aber Achtung: das Zurückholen mit R 〉 muß in der gleichen Zeile vor dem betätigen der RETURN-Taste geschehen. Sonst ist die Zahl verloren, da FORTH diesen RETURN-Stapel für die Programmausführung benötigt.

Auf weitere Möglichkeiten, den Stapel zu verändern wird später noch eingegangen werden. Bevor wir die bisherigen Befehle in Programmbeispielen anwenden können, müssen wir noch zeigen, wie man in FORTH seine eigenen Befehle und Worte definiert.

2.6. Definition eigener Worte

Die Möglichkeit schon definierte Worte aus dem Wörterbuch in einem neuen Wort zusammenzufassen, ist eine der mächtigsten Eigenschaften von FORTH. Eine Wortdefinition beginnt mit einem : gefolgt von

einem Leerzeichen. Alle folgenden Zeichen bis zum nächsten Leerzeichen bilden das neue Wort. Dann folgt die Angabe der schon definierten Worte. Abgeschlossen wird diese Definition mit einem ;

: XXX (n1 n2 n3 --> n1 n2 n3)

Beginn der Definition von XXX

; (n1 n2 n3 --> n1 n2 n3)

Ende der Definition

Als erstes Beispiel wollen wir die Ausgabeanweisung . durch das Wort DRUCKE ersetzen. Die FORTH-Zeile dafür lautet:

```
: DRUCKE . ;   OK
```

Dieses Wort wird, wenn die Eingabe mit CR beendet wird, ins Wörterbuch eingetragen. Wenn es aufgerufen wird, führt es die Ausgabeanweisung . aus.

Anstatt 1 2 3 . 3 OK

können wir jetzt 1 2 3 DRUCKE

schreiben und der Computer wird mit

3 OK

antworten.

In einem zweiten Beispiel wollen wir den Term.

$$X^2 + 4X + 5$$

für einige Werte von X als ganze Zahl auswerten.

Das Programm für X = 3 lautet dann

```
3 DUP DUP * SWAP 4 * + 5 + . 26  OK
```

Dieses Programm soll zu einem einzigen Wort zusammengefasst werden. Nur der eingegebene X-Wert soll jedesmal verschieden sein.
Zuerst fassen wir die zwei DUP in einem einzigen Wort zusammen.

```
: DD DUP DUP ;  OK
```

und schreiben dann unser Programm, für das wir den Namen # wählen

```
: # DD * SWAP 4 * + 5 . ;
# NOT UNIQUE   OK
```

Nach dem CR erhalten wir allerdings eine Warnung:

```
# NAME NOT UNIQUE OK
```

oder vielleicht auch

```
# ISN'T UNIQUE OK
```

Das heißt, wir haben jetzt ein Wort definiert, das schon ins Wörterbuch eingetragen ist. Der Rechner antwortet trotzdem mit OK, denn wenn ab sofort das # aufgerufen wird, so wird die gerade eingegebene Programmzeile ausgeführt, und nicht der früher definierte Programmablauf für #.

Wir können jetzt also eingeben:

```
1 #  10  OK

-1 #  2  OK

3 #  26  OK                    usw.
```

Zum besseren Verständnis ist der Programmablauf im Stapel in Abbildung 2.2 gezeigt.

STAPEL					TOS	EINGABE
					3	3
			3	3	3	DD
				3	9	*
				9	3	SWAP
			9	3	4	4
				9	12	*
					21	+
				21	5	5
					26	+

Abbildung 2.2
Berechnung des Terms $X^2 + 4X + 5$ mit $X = 3$

Obwohl sich die beiden Sprachen kaum vergleichen lassen, ist in Abbildung 2.3 das BASIC-Programm und das FORTH-Programm zur Berechnung dieses Ausdrucks gegenüber gestellt.

```
BASIC PROGRAMM:        FORTH PROGRAMM

10 INPUT X             : DD DUP DUP ;
20 Y=X*X+4*X+5         : # DD * SWAP 4 * + 5 + . ;
30 PRINT Y
```

Abbildung 2.3
Vergleich BASIC-Programm mit FORTH-Programm

Natürlich möchte man feststellen, ob diese Namen ins Wörterbuch übernommen sind und welche Namen sonst dort eingetragen sind. Das Wort zu Ausgabe des Wörterbuchs heißt VLIST.

Mit VLIST CR wird das ganze Wörterbuch auf den Bildschirm ausgegeben. Die obersten beiden Einträge sind unsere gerade definierten Wort # und DD, wobei die zuletzt gemachte Definition # an oberster Stelle im Wörterbuchstapel steht.

Das Löschen von Einträgen geschieht mit dem Wort FORGET. Mit FORGET # wird nur das Wort # gelöscht, mit FORGET DD die beiden neuen Worte # und DD .

FORGET XXX (n --> n)

Löscht XXX und alle darüberliegenden Worte.

Beim Schreiben und Testen von FORTH-Programmen muß man sich immer vor Augen halten, daß das Wörterbuch ebenfalls ein Stapel ist. Hätten wir zum Beispiel bei der Definition des Wortes # einen Fehler gemacht, so wäre mit FORGET # nur diese Zeile gelöscht worden. Danach hätte man nur diese Zeile neu eingeben müssen.

Wenn man einen Fehler in DD gemacht hat und löscht diese Zeile mit FORGET DD, so sind beide neuen Zeilen verschwunden und müssen neu eingegeben werden. Deshalb ist die direkte Eingabe von FORTH Worten nur bei ganz kurzen Programmen sinnvoll. In allen anderen Fällen müssen die Programme über den Editor eingegeben werden (siehe Kapitel 5).

Bei einem Fehler in dem Wort DD hätte auch eine Neudefinition ohne vorheriges Löschen der alten Definition nichts gebracht, denn die Reihenfolge im Wörterbuch wäre dann

DD # DD VLIST ....

Das Wort # enthält aber immer noch die hinter ihm stehende Definition von DD und nicht die des vor ihm stehenden Wortes DD.

Dazu noch ein Beispiel. Wir definieren der Reihe nach:

```
: EINS 1 . ;   OK

: ZWEI 2 . ;   OK

: DREI 3 . ;   OK

: DREI ZWEI ;
DREI NOT UNIQUE    OK
```

Beim Aufruf von DREI erhalten wir 2 OK . Nach FORGET DREI ergibt der Aufruf von DREI 3 OK . Wenn wir jetzt FORGET ZWEI eingeben und danach DREI aufrufen, kommt die Meldung DREI ? . Nach diesem FORGET ZWEI ist nur noch der Eintrag EINS erhalten geblieben.

Da die Wörter in FORTH beliebige Buchstaben- und Zeichenfolgen sein können, ist auch möglich, z. B. Rechenzeichen zu verändern. Wer lieber den : statt des / als Divisionszeichen haben möchte, der definiert einfach

```
: : / ;
```

Damit ist aber das Eröffnungszeichen für Wortdefinitionen verloren gegangen.

Besonders trickreich ist es, die Bedeutung von Rechenzeichen zu verändern. Mit

```
: + - ;
```

wird aus der Addition eine Subtraktion. Somit erhält man für

```
4 2 + .
```

nicht	6 OK
sondern	2 OK

Mit diesem Wörterbuch lassen sich natürlich sehr leicht Sprachübersetzer programmieren. Die folgende Zeile

```
: LEECH ." BLUTEGEL" ;
```

weist dem englischen Wort leech das deutsche Wort Blutegel zu.

Für die Programmierung eines Sprachübersetzers sind also keinerlei Programmierkenntnisse notwendig, da FORTH dieses Wörterbuchkonzept schon beinhaltet. Man kann sich leicht mehrere solcher Wörterbücher anlegen, diese auf Diskette speichern und wenn sie gebraucht

werden, wieder in den Rechner einlesen. Nach diesem Prinzip arbeiten die meisten käuflichen maschinellen Sprachübersetzer.

Sogar einfache Sätze lassen sich damit bilden. Dazu folgende Beispiele:

```
: I ." ICH " ;
: AM ." BIN " ;
: HERE ." HIER " ;
```

Der Aufruf von:

I AM HERE ergibt ICH BIN HIER OK

Natürlich erfolgt keine Syntaxüberprüfung. Denn die Feststellung

HERE I AM ergibt HIER ICH BIN OK

Dieses schöne Konzept eines Sprachübersetzers arbeitet leider nicht mit allen FORTH Versionen. Nur FORTH-79 und die ihm ähnlichen Dialekte erlauben die Definition von Worten bis zu 31 Zeichen. Andere FORTH-Versionen tragen ins Wörterbuch nur die ersten drei Zeichen und die Anzahl der Zeichen ein. Das obige Beispiel mit

```
: LEECH ." BLUTEGEL " ;
```

funktioniert hier auch, aber andere Worte wie LEEXX oder LEEST führen dann ebenfalls zur gleichen Übersetzung.

Eine andere Art, einen Sprachübersetzer zu programmieren, wird später noch gezeigt werden.

Verwendet man solch einen FORTH-Dialekt, so ist es sinnvoll, bei der Definition von Worten, die eine ähnliche Funktion haben und deshalb ähnlich lauten, diese durch das erste Zeichen zu unterscheiden, wie z. B.

2DUP 3DUP oder 2SPACE 4SPACE

2.7 Konstante und Variable

Obwohl die meisten Zahlenwerte in FORTH über den Stapel verarbeitet werden, so kann man trotzdem auch Konstante und Variable definieren.

Die Definition einer Konstanten ist einfach. Durch

```
10 CONSTANT Y  OK
```

wird im Wörterbuch eine Konstante Y eingetragen, die den Zahlenwert 10 hat. Dieser Wert kann dann im Programm aufgerufen werden.

Beispiel:

```
12 Y + .  22  OK
```

CONSTANT ⟨NAME⟩ (n1 n2 n3 -- ⟩ n1 n2)

Der Wert der Konstanten ⟨NAME⟩ ist n3

Das gleiche Ergebnis hätte man auch mit der Definition

```
: Y 12 ;   OK
```

erreicht. Die Ausführungszeit von Programmen mit Konstantendefinition ist jedoch wesentlich kürzer.

Auf die gleiche Weise wird eine Variable definiert:

```
10 VARIABLE X  OK
```

VARIABLE ⟨NAME⟩ (n1 n2 n3 -- ⟩ n1 n2)

Der Anfangswert der Variablen ⟨NAME⟩ ist n3

Zwischen einer Konstanten und einer Variablen besteht ein sehr wesentlicher Unterschied. Eine Konstante behält im Programm ihren Wert. Dieser wird beim Aufruf der Konstanten auf dem Stapel abgelegt. Beim

Aufruf einer Variablen wird dagegen nicht der Wert der Variablen, sondern die Adresse dieser Variablen auf dem Stapel abgelegt. Zum Ändern des Wertes einer Variablen benötigen wir noch zwei weitere Befehle, nähmlich den Befehl: Wert holen (LOAD) und den Befehl: Wert wegspeichern (STORE).

Für den STORE-Befehl wird das ! verwendet.

! (n1 n2 adr --> n1)

Der Wert n2 ist nach addr gespeichert worden.

Für den LOAD-Befehl wird das @ Zeichen verwendet.

@ (n1 n2 adr --> n1 n2 n)

Der Inhalt von adr ist auf dem Stapel abgelegt worden.

Beispiel:

```
10 VARIABLE X CR   OK
X @ . 0   OK
2 X !   OK
X @ . 2   OK
```

Wenn wir statt X @ . nur X . eingeben, erhalten wir die Adresse bei welcher der Wert der Variablen gespeichert ist.

```
X . CR   8071 OK
```

Diese Adresse wird jedoch, je nach verwendeten Rechner und angelegten Wörterbuch, immer verschieden sein. Den Wert der Variablen X lässt sich dann auch durch

```
15 8071 ! CR   OK
```

ändern.

Mit

X @ . CR

erhalten wir dann

15 OK

Die Zeichenfolge @ . ist zu dem Zeichen ? zusammengefasst worden. Mit

X ? CR

erhält man ebenfalls

15 OK

Auf ähnliche Weise kann man auch den Wert einer schon vereinbarten Konstanten ändern. Die Adresse einer Konstanten erhält man mit dem WORT ' (Einzelapostroph).

' ⟨ NAME ⟩ (n1 n2 -- ⟩ n1 n2 adr)

Die Adresse der Konstanten ⟨ NAME ⟩ wird auf dem Stapel geholt.

Beispiel:

```
10 CONSTANT W  OK

W . 10  OK
20 ' W !  OK
W . 20  OK
```

In der dritten Zeile dieses Programms wird die Zahl 20 auf den Stapel geschrieben und dann mit ' W die Adresse der Konstanten W geholt. Der folgende ! Befehl speichert die Zahl 20 in diese Adresse. Damit ist der Wert der Konstanten verändert.

Die Befehle ! STORE und @ LOAD lassen sich also auf alle Speicher zellen anwenden. Für diejenigen, die mal wieder einen Vergleich mi BASIC haben möchten:

	n adr !	entspricht	POKE adr, n
und	adr @	entspricht	PEEK (adr)

n ist allerdings eine 16 Bit Zahl. Will man dagegen nur ein einzelne Byte, also eine Dezimalzahl zwischen 0 und 255 abspeichern, so ver wendet man das Wort C! . Mit C@ wird ein Byte auf den Stape geholt, dabei werden die oberen 8 Bit zu Null gesetzt.

Verwendet man in BASIC die Befehle PEEK und POKE, so muß man den Wert und die Adresse immer dezimal angeben. In FORTH kann man sich dagegen die Zahlenbasis, in der man arbeiten möchte frei wählen. Bei der Initialisierung von FORTH wird immer das Rechnen zur Basis 10, also Dezimalrechnung, angenommen. Das Um schalten zum Rechnen mit der Basis 16, Hexadezimalrechnung ge schieht mit dem Wort HEX.

HEX (n1 n2 n3 --> n1 n2 n3)

Hexadezimalrechnung

Das Rücksetzen in die Dezimalrechnung geschieht mit DECIMAL

DECIMAL (n1 n2 n3 --> n1 n2 n3)

Dezimalrechnung

Beispiel:

```
DECIMAL 100 HEX . 64  OK
```

Alle Zahlen, die jetzt eingegeben werden, werden als Hexadezimal zahlen interpretiert.

```
FFFF DECIMAL . -1  OK
```

Damit haben wir aber auch die Bestätigung, daß unsere Zahlen im 2-er Komplement dargestellt werden. Will man also Hexadezimalzahlen direkt in den Speicher schreiben, so schaltet man um in Hexadezimal-Darstellung und speichert die Werte bei den gegebenen Adressen ab. Man muß dabei allerdings beachten, daß dies immer 16-Bit Zahlen sind, die im Speicher 2 Byte belegen. Sollen nur einzelne Bytes verwendet werden, so verwendet man die C! und C @ Befehle.

```
HEX

A9 6000 C!

FF 6001 C!

: .C C@ . ;

6000 .C  A9  OK
```

In der Mitte dieser Programmfolge haben wir uns eine neue Ausgabe definiert, die uns den Inhalt einer Speicherzelle auf den Bildschirm ausgibt. Mit

```
6000 DECIMAL . 24576  OK
```

erhalten wir den Dezimalwert dieser Adresse und mit

```
24576 .C  169  OK
```

den Inhalt dieser Zelle als Dezimalzahl.

Bei den Zahlenumwandlungen wird eine intern vereinbarte Variable BASE benutzt, deren Wert der augenblicklich verwendeten Basis zur Darstellung von Zahlen entspricht. Ändert man diesen Wert, so kann man auch mit anderen Zahlensystemen arbeiten. Mit

```
: BIN 2 BASE ! ;
```

definieren wir ein binäres Zahlensystem und mit

```
: TRI 3 BASE ! ;
```

ein trinäres Zahlensystem, das nur die Zahlen 0, 1, 2, kennt.

Die folgenden beiden Definitionen geben eine Dezimalzahl als Binä oder Trinärzahl auf den Bildschirm aus.

```
: .BIN BIN . DECIMAL ;

: .TRI TRI . DECIMAL ;

120 .BIN 1111000  OK

140 .TRI 12012  OK
```

In den beiden obigen Zeilen tritt zweimal die gleiche Wortfolge BAS ! . DECIMAL auf. Definiert man hierfür ein neues Wort, so kann ma die Schreibarbeit verkürzen.

```
: >> BASE ! .DECIMAL ;

: .BIN >> ;

: .TRI >> ;
```

Dieses Beispiel zeigt schon, das FORTH Programme sehr kurz, aber d durch nicht unbedingt lesbarer werden.

2.8. Übungen und Beispiel

Es folgen nun einige Übungen zu bisherigen Inhalt. Außerdem soll a einem Beispiel gezeigt werden, was man mit den bisherigen Kenn nissen schon programmieren kann.

[illegible] einige Aufgaben zum Umsetzen von algebrischen Ausdrücken [illegible] postfix Notation.

ÜBUNG 1

a) $1 + 5 * 7$

b) $\dfrac{6 * 4 - 3}{7} - 2$

c) $5^2 \quad 4^2$

d) $(2 * (5 - 3) * (6 - 2)) + 4$

e) $\dfrac{20}{(3 + 2) * 4}$

In der nächsten Übung wird die Wortfolge gesucht, welche die folgenden Veränderungen am Stapel durchführen. Das oberste Element des Stapels steht immer am weitesten rechts.

ÜBUNG 2

X Y Z --> X Z Y
X Y Z --> Y Z X
X Y Z --> Y X Z
X Y Z --> Z Y X
X Y Z --> X Y Z Y Z
X Y Z --> X Y X

In der nächsten Übung sollen die Definitionen der folgenden Worte bestimmt werden:

ÜBUNG 3

a) 2DUP (n1 n2 --> n1 n2 n1 n2)

b) 2DROP (n1 n2 n3 --> n1)

c) 2SWAP (n1 n2 n3 n4 --> n3 n4 n1 n2)

d) 2OVER (n1 n2 n3 n4 --> n1 n2 n3 n4 n1 n2)

e) NIP (n1 n2 n3 --> n1 n3)

f) 4REV (n1 n2 n3 n4 --> n4 n3 n2 n1)

In der nächsten Übung sollen algebraische Ausdrücke berechnet werde
Auf dem Stapel sind die Zahlen X Y Z gespeichert, mit Z im TOS. G
sucht sind die Wortfolgen.

ÜBUNG 4

a) $X^2 + XY - Z$

b) $X^2 - XY + Z$

c) $X^2 + XY + Z^2$

d) $X^2 - XY - Z^2$

und mit YZ auf dem Stapel:

e) $Y + Y/Z + Z$

f) $Y + Y/Z - Z$

g) $Y - Y/Z - Z$

In Übung 5 sind Wortfolgen gegeben. Welche Zahl ist zum Schluß a
dem Stapel ? Bei Beginn der Wortfolge ist der Stapel leer.

ÜBUNG 5

a) 3 DUP DUP 2 + - -

b) 1 2 3 ROT OVER + +

c) 2 DUP * 4 + 3 DUP OVER DROP +

Programmbeispiel 1: Kalorienberechnung

Zur Berechnung einer täglich verbrauchten Kalorien, bzw. Joulemenge soll die Eingabe in den Rechner in der folgenden Form geschehen:

```
BROT    3 SCHEIBEN
KAESE   100 GR
MILCH   1 GLAS          usw.
```

Die Aufsummierung soll auf dem Stapel erfolgen und durch Aufruf des Wortes Summe ausgegeben werden. Diese Zahl soll jedoch auf dem Stapel erhalten bleiben.

Die Programmierung dieses Beispiels ist recht einfach (Abbildung 2.81). Wir ordnen den Nahrungsmitteln die Zahl der Kalorien in einer Messeinheit zu. Zum Beispiel:

```
: APFELSAFT 104 ;
```

Die Einheit ist hier ein Glas, oder

```
: BIERWURST 36 ;
```

die Einheit ist hier 10 Gramm.

In unserem Beispiel sind auf diese Weise einige Nahrungsmittel und Getränke aufgeführt.
Die Berechnung wird durch die Maßeinheiten ausgeführt. Betrachten wir folgende Eingabe

```
LIMONADE 3 GLAESER
```

Vor dem Wort LIMONADE ist die bisherige Summe im TOS. Durch das Wort LIMONADE wird die Zahl 110 und durch das Wort 3 die Zahl 3 auf den Stapel gelegt. Durch das Wort Gläser wird zuerst das Produkt 3 * 110 gebildet und das Ergebniss zur bisherigen Summe hinzuaddiert.

Vor der ersten Eingabe muß eine Null auf den Stapel geschrieben

werden, als Ausgangswert für die bisherige Summe. In dieser Form ha das Programm den Nachteil, daß bei einer falschen Eingabe der Stape gelöscht wird. Man wird also die Aufsummierung in einer Variable SUMME durchführen.

```
( KOMMENTAR ZUR KALORIEN BERECHNUNG
  EINHEIT GLAS : APFELSAFT,PILS,BIER,LIMONADE,
                 COCA-COLA,CHAMPAGNER.
  EINHEIT 10 GRAMM: BIERWURST,BLUTWURST,BUTTER
                 EDAMER.

  EINHEIT STUECK:BROETCHEN,CAMEMBERT.

  EINHEIT SCHEIBE: BROT.

  EINGABE IN DER FORM:

  NAHRUNGSMITTEL ANZAHL MASSEINHEIT.


( KALORIENWERTE )
: APFELSAFT 104 ; : PILS 146 ; : BIER 255 ;
: BIERWURST 36 ;  : BLUTWURST 42 ;
: LIMONADE 110 ;
: BROETCHEN  140 ; : BROT 100 ; : BUTTER 78 ;
: CAMEMBERT 200 ; : CHAMPAGNER 67 ;
: COCA-COLA 88 ;
: EDAMER 34 ;


( MASS-EINHEITEN )
: GLAS DROP + ; : GLAESER * + ;
: GRAMM 10 / * + ;
: SCHEIBE DROP + ; : SCHEIBEN * + ;
: STUECK * + ;


: SUMME DUP . ;
APFELSAFT 1 GLAS  OK
BROETCHEN 2 STUECK  OK
EDAMER 200 GRAMM  OK
BIERWURST 150 GRAMM  OK
SUMME 1604  OK
```

Abbildung 2.81
Kalorienberechnun

Programmbeispiel 2: Morsen

Die meisten Heimrechner besitzen einen Lautsprecher, über den durch ein Steuerzeichen ein Ton ausgegeben werden kann. Beim Apple II wird zum Beispiel durch

HEX 87 EMIT

ein kurzer Ton ausgegeben.
Im Morseprogramm wird nun durch das Wort . ein kurzer Ton, durch das Wort – ein längerer Ton und durch das Wort / eine Pause definiert.

Dann kann man im Morsecode ganze Worte (Beispiel SOS) oder einzelne Buchstaben angeben.

Durch Aufruf der Worte oder durch Eingeben einer Buchstabenfolge getrennt durch Leerzeichen wird der Morsecode ausgegeben.
Das Programm zeigt Abbildung 2.82.

```
( MORSE PROGRAMM )    HEX
: / 500 0 DO LOOP ;
: - 3 0 DO 87 EMIT LOOP / ;
: . 87 EMIT / ;
: SOS . . . / - - - / . . . ;
: E . / ; : I . . / ; : S . . . / ;
: H . . . . / ;
: T - / ; : M - - / ; : O - - - / ;
```

```
DECIMAL
 OK
```

Abbildung 2.82
Morseprogramm

2.9. Zusammenstellung der bisherigen Definitionen

WORT	STAPEL VORHER	STAPEL NACHER
.	(n1 n2 n3 -->	n1 n2)

Ausgabe von n3 auf Bildschirm

." XXX "	(n -->	n)

Ausgabe von XXX als Text.

SPACE	(n -->	n)

Ausgabe eines Leerzeichens.

SPACES	(n1 n2 n3 -->	n1 n2)

Ausgabe von n3 Leerzeichen.

CR	(n -->	n)

Beginn einer neuen Zeile

.R	(n1 n2 n3 -->	n1 n2)

Tabulator wird auf n3 Stellen gesetzt. Die Zahl wird rechtsbünd ausgegeben.

EMIT	(n1 n2 c -->	n1 n2)

c wird als 7 bit ASCII-Zeichen auf den Bildschirm ausgegeben.

+ (n1 n2 n3 --> n1 n)

Addition n = n2 + n3

- (n1 n2 n3 --> n1 n)

Subtraktion n = n2 – n3

* (n1 n2 n3 --> n1 n)

Multiplikation n = n2 * n3

/ (n1 n2 n3 --> n1 n)

Division n = n2 / n3

MOD (n1 n2 n3 --> n1 n)

n Rest der Division n2 / n3

/MOD (n1 n2 n3 --> n1 n4 n)

n n2 / n3, n4 = Rest

MAX (n1 n2 n3 --> n1 n)

n ist die größere der beiden Zahlen n2, n3

MIN (n1 n2 n3 --> n1 n)

n ist die kleinere der beiden Zahlen n2, n3

ABS (n1 n2 n3 --> n1 n2 n)

n ist der Absolutwert von n3

MINUS (n1 n2 n3 --> n1 n2 –n3)

n3 wird negativ

AND (n1 n2 n3 --> n1 n)

n = n2 & n3

OR (n1 n2 n3 --> n1 n)

n = n2 V n3

XOR (n1 n2 n3 --> n1 n)

n = n2 ∧ n3

DUP (n1 n2 n3 --> n1 n2 n3 n3)

Duplizieren des obersten Elementes auf dem Stapel.

SWAP (n1 n2 n3 --> n1 n3 n2)

Vertauschen der beiden obersten Elemente auf dem Stapel.

OVER (n1 n2 n3 --> n1 n2 n3 n2)

2-tes Element des Stapels duplizieren und auf den Stapel leg

ROT (n1 n2 n3 --> n2 n3 n1)

Die obersten 3 Elemente des Stapel zyklisch vertauschen.

DROP (n1 n2 n3 --> n1 n2)

Oberstes Element löschen

[illegible] (n1 n2 n3 --> n1 n2 n4)

[illegible] /9) n4 ist n3-tes Element des Stapels, dupliziert.

[illegible] (n1 n2 n3 --> n1 n2 n4)

[illegible] n3.tes Element des Stapels. Alle anderen eine Stelle tiefer [illegible].

[illegible] (n1 n2 n3 --> n1 n2)

[illegible] wird auf den Returnstapel geschrieben.

[illegible] (n1 n2 --> n1 n2 n3)

[illegible] wird aus dem Returnstapel zurückgeholt.

[illegible] (n1 n2 --> n1 n2 n3)

Das oberste Element n3 des Returnstapels wird auf den Rechenstapel kopiert.

XXX (n --> n)

Beginn der Definition des Wortes XXX.

(n --> n)

Ende der Wortdefinition.

VLIST (n --> n)

Ausgabe des augenblicklichen Wörterbuchs.

FORGET XXX (n --> n)

Löschen der Definition XXX einschließlich aller später gemacht Definitionen.

CONSTANT ‹NAME› (n1 n2 n3 --> n1 n2)

Definition einer Konstanten ‹NAME›, deren Wert n3 ist.

VARIABLE ‹NAME› (n1 n2 n3 --> n1 n2)

Definition einer Variablen ‹NAME›, deren Wert n3 ist.

! (n1 n2 adr --> n1)

Der Wert n2 wird nach adr gespeichert.

@ (n1 n2 adr --> n1 n2 n)

n ist der Inhalt der Speicherzelle adr .

' ‹NAME› (n1 n2 --> n1 n2 adr)

Die Adresse der Konstanten ‹NAME› wird auf den Stapel geholt.

? (n1 n2 adr --> n1 n2)

entspricht @ .

C! (n1 b adr --> n1)

8-bit Byte wird nach adr gespeichert.

C@ (n1 n2 addr --> n1 n2 b)

8-bit Byte wird aus adr geholt.

HEX (n --> n)

Zahlendarstellung Hexadezimal.

DECIMAL (n --> n)

Zahlendarstellung Dezimal.

BASE (n --> n)

Variable, enthält den augenblicklichen Wert der Zahlenbasis.

Auch Sie brauchen ELCOMP!

Jahresabonnement **69,00 DM** incl. Mwst. und Versand.

Zurückliegende Hefte zu Originalpreisen noch verfügbar.

ELCOMP

HOFACKER-VERLAG
Ing. W. Hofacker GmbH
Tegernseer Straße 18
D-8150 Holzkirchen/Obb.

Die Fachzeitschrift für MICROCOMPUTER
Eine unentbehrliche Informationsquelle für alle Elektroniker

Microcomputer-Anwendungsbeispiele
Künstliche Intelligenz
Block-Strukturierte Programme
Datenverarbeitung im Kleinbetrieb
Club-Neuheiten
Computer und Kunst
Musik mit dem Computer
Monitore für 8080, 6800, 6502, Z 80, SC/MP, 2650, 1802
Eigenbau-Computersysteme
Interface-Techniken
Microcomputer KITs
Neue Produkte
Betriebssysteme für Floppys
Programmiertechniken
Software-Quellen
Programmierbeispiele
Soziale Aspekte der Microcomputertechnik
Technologische Neuheiten
Anwendungen in der Meß- und Regeltechnik
Anwendungen bei Funk-Amateuren

3 Programmschleifen und Verzweigungen

3. Programmschleifen und Verzweigungen

Die bisher beschriebenen Worte sind ausschließlich Befehle, die sofort ausgeführt werden, aber keinerlei Steuerung des Programmablaufs bewirken. In FORTH ist, genau wie in PASCAL, kein GOTO-Befehl, also ein unbedingter Sprungbefehl vorhanden. Damit sind FORTH-Programme also strukturiert und benutzen zu Programmverzweigungen Schleifenbedildungen der Form BEGIN . . . UNTIL oder BEGIN . . . WHILE . . .REPEAT oder Abfragen mit IF. . . THEN. Diese Befehlsfolgen können aber nicht direkt ausgeführt werden, sondern müssen innerhalb einer Wortdefinition verarbeitet werden. Der Programmierer muß dafür Sorge tragen, daß diese Schleifen auch terminieren und somit keine unendlichen Schleifen auftreten.

3.1. DO . . . LOOP

Mit der Definition der DO . . . LOOP wird eine feststehende Zahl von Schleifendurchläufen ausgeführt.

DO . . . LOOP (n1 n2 n3 --> n1)

Schleife wird von n3 bis n2 in Einerschritten durchlaufen. n2 > n3.

Der Anfangswert für die Schleife ist n3, der Endwert der Schleife ist n2. Dazwischen wird der Zählparameter immer um Eins erhöht. Der Endwert muß also immer größer sein als der Anfangswert. Bei jedem Schleifendurchlauf wird geprüft, ob der Zählerparameter gleich oder größer als der Endwert n2 ist. Ist dies der Fall, so wird die Schleifen-

bildung abgebrochen. Diese Abfrage erfolgt aber erst am Ende einer Schleife, sodaß jede Schleife mindestens einmal durchlaufen wird, auch dann, wenn der Anfangswert größer als der Endwert ist. Der Zählparameter ist als I definiert und kann innerhalb einer Schleife auf den Stapel geholt werden.

I (n1 n2 n3 --> n1 n2 n3 n)

n Zählparameter der Schleife.

Beispiele:
Ausgabe der Zahlen 0 bis 9 auf den Bildschirm.

```
: T 10 0 DO I . LOOP ;  OK
T 0 1 2 3 4 5 6 7 8 9  OK
```

Vertauscht man die beiden Zahlen 10 und 0 in der obigen Definition, so wird die Schleife einmal durchlaufen.

```
: T 0 10 DO I . LOOP ;  OK
T 10  OK
```

Man muß also bei der Programmierung von DO . . . LOOP Schleifen darauf achten, daß diese auf jedenfall mindestens einmal durchlaufen werden.
Im nächsten Beispiel soll die Verwendung von geschachtelten Schleifen gezeigt werden.

Die Zahlen von 0 bis 9 sollen in einer Dreiecksmatrix ausgegeben werden.
Dafür definieren wir 2 Schleifen IS und AS. IS ist die innere und AS die äußere Schleife.

```
: IS 0 DO I . LOOP ;
: AS 0 DO CR 1 - DUP IS LOOP ;
```

Der Aufruf 11 10 AS CR druckt dann das folgende Zahlenbild aus (Abbildung 3.1.).

```
11 10 AS
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6
0 1 2 3 4 5
0 1 2 3 4
0 1 2 3
0 1 2
0 1
0  OK
```

Abbildung 3.1
Dreiecksmatrix

Der Aufruf von 10 IS druckt die Zahlen von 0 bis 9 aus. In der Schleife AS wird aber, vevor die Schleife IS ausgeführt wird, von der obersten Zahl im Stapel Eins abgezogen. Der erste Parameter ist somit 11, die Schleife AS wird 10 mal durchlaufen.

Eine andere Dreiecksmatrix erhält man z. B. mit 16 15 AS

In der Definition der äußeren Schleife wird ein neues, im Wörterbuch vorhandenes Wort CR verwendet. Dies bewirkt die Ausgabe einer neuen Zeile auf den Bildschirm.

Mit einer DO . . . LOOP Schleife kann man auch zeigen, daß FORTH eine sehr schnelle Sprache ist. Die Schleife

```
: T 10000 0 DO LOOP ;
```

wird in weniger als 3 Sekunden ausgeführt (ATARI 800).

Nicht immer kann man Schleifen verwenden, die den Zählparameter immer in Einerschritten erhöhen. Für andere Schrittweiten verwendet man die DO . . . + LOOP. Dabei wird als Increment der Zahlenwert verwendet, der vor dem Schleifenende auf dem Stapel ist. Mit

```
DO 5 +LOOP
```

erfolgt eine Erhöhung des Zählparameters um 5 oder mit

```
DO -2 +LOOP
```

wird der Zählparameter laufend um 2 erniedrigt. Der Abbruch der Schleife erfolgt, wenn der neue Index größer oder gleich der oberen Grenze oder wenn der neue Index gleich oder kleiner als die untere Grenze ist (kann bei einzelnen FORTH-Versionen abweichen).

```
: T DO I . 2 +LOOP ;   OK
10 0 T  0 2 4 6 8  OK

: M DO I . -3 +LOOP ;   OK

3 12 M 12 9 6 3  OK
```

+LOOP (n1 n2 n3 -- > n1 n2)

Der Zählparameter wird um n3 verändert.

3.2. Vergleichsbefehle

Wird ein Programm an einer Stelle verzweigt, so geschieht dies im allgemeinen durch einen Vergleich. Je nach Aussage dieses Vergleiches wird z. B. ein Programm weitergeführt oder der vorangegangene Teil nochmals ausgeführt.

Als Vergleiche stehen in FORTH folgende Worte zur Verfügung:

< (n1 n2 n3 -- > n1 f)

f = 1 wenn n2 < n3

> (n1 n2 n3 -- > n1 f)

f = 1 wenn n2 > n3

= (n1 n2 n3 -- > n1 f)

f = 1 wenn n2 = n3

0< (n1 n2 n3 -- > n1 n2 f)

f = 1 wenn n3 < 0

0= (n1 n2 n3 -- > n1 n2 f)

f = 1 wenn n3 = 0

Als Ergebnis eines Vergleiches wird immer auf dem Stapel eine Boolsche Zahl f abgelegt, die eins ist, wenn die Vergleichsbedingung erfüllt und Null, wenn der Vergleich nicht erfüllt ist.

Beispiele:

```
2 3 < . 1 OK
3 2 < . 0 OK
3 2 = . 0 OK
-2 0< . 1 OK
```

Bei einem Vergleich zwischen den beiden obersten Zahlen auf dem Stapel werden diese beiden Zahlen durch das Ergebnis des Vergleichs ersetzt. Will man diese Zahlen für die weitere Rechnung verwenden, so kann man durch

OVER OVER

beide Zahlen in ihrer Reihenfolge auf dem Stapel kopieren.

2 4 OVER OVER . . . .

druckt 4 2 4 2 OK aus.

3.3. IF . . . ELSE . . . ENDIF

Dies ist eine der einfachsten Programmverzweigungen in FORTH.

Die Definitionen sind:

IF ⟨WORTE⟩ ENDIF

IF ⟨WORTE⟩ THEN

IF (n1 n2 f -- > n1 n2)

Die ⟨WORTE⟩ zwischen IF und ENDIF werden nur ausgeführt, wenn f = 1 ist.

IF ⟨WORTE1⟩ ELSE ⟨WORTE2⟩ END'F

IF ⟨WORTE1⟩ ELSE ⟨WORTE2⟩ THEN

IF (n1 n2 f -- > n1 n2)

Die ⟨WORTE1⟩ werden ausgeführt, wenn f = 1 ist. Ist f = 0 so werden die ⟨WORTE2⟩ ausgeführt.

Statt des Wortes ENDIF wird in einigen FORTH-Versionen das Wort THEN verwendet.

Beispiel:

Das Wort VERGL vergleicht die beiden obersten Zahlen im Stapel und

durckt das Ergebnis in Worten aus.

```
: VERGL < IF ." KLEINER" ELSE ." GROESSER" THEN ;
```

```
2 3 VERGL KLEINER OK

3 2 VERGL GROESSER OK
```

3.4. BEGIN . . . UNTIL

Die BEGIN . . . UNTIL Schleife hat folgenden Aufbau:

BEGIN ⟨WORTE⟩ ⟨BEDINGUNG⟩ UNTIL

Die Wortfolge zwischen BEGIN und UNTIL wird solange durchlaufen, bis die vor UNTIL ausgeführte Bedingung erfüllt ist. Die Ausführung der Bedinung legt die Boolsche Variable f auf dem Stapel ab. Diese Variable wird durch das Wort UNTIL wieder vom Stapel genommen. War f = Eins, dann wird die Schleife verlassen, war f = Null, so wird die Schleife wiederholt.

Beispiel:
Die Schleifenbildung mit BEGIN . . . UNTIL soll dazu verwendet werden, die Zahlen von 1 bis 100 zusammenzuzählen.

```
0 VARIABLE X

: INCX X @ 1 + DUP X ! ;

: SUM 0 BEGIN INCX + X @ 100 = UNTIL . ;

SUM 5050  OK
```

Abbildung 3.2
Bildung der Summe von 1 bis 100

Das Programm besteht aus zwei Worten INCX und SUM. In INCX wird die Variable X immer um Eins erhöht und der augenblickliche Wert auf den Stapel geschrieben. Dann wird X wieder weggespeichert.

In SUM wird zuerst eine Null auf den Stapel geschrieben und dann INCX aufgerufen. Die bisherige Summe und der neue X Wert werden addiert. Dann wird X wieder geholt und mit 100 auf Gleichheit überprüft. Ist diese Bedingung erfüllt, wird die Schleife verlassen und das oberste Element des Stapel, die aufaddierte Summe, ausgegeben und somit das Programm beendet.

Beispiel:

Das Tastenfeld wird laufend abgefragt, ob die Taste K gedrückt wurde. Dazu wird das Wort KEY verwendet, das dem Befehl GET$ in BASIC entspricht. Das Programm wartet auf einen Tastendruck und legt das entsprechende ASCII-Zeichen auf dem Stapel ab.

KEY (n1 n2 n3 --> n1 n2 n3 c)

c = 7 bit ASCII-Zeichen.

```
: INKEY BEGIN KEY 75 = UNTIL ." K" ;  OK

INKEY K OK
```

Das Wort INKEY wird mit der Ausgabe des Buchstabens K nur dann verlassen, wenn die Taste K gedrückt wurde.

3.5. BEGIN ... WHILE ... REPEAT

Der Aufbau dieser Schleife weist Unterscheide zu den bisherigen Schleifen auf. Der Programmablauf der Schleife ist folgender:

BEGIN ⟨WORTE1⟩ ⟨BEDINGUNG⟩ WHILE ⟨WORTE2⟩ REPEAT ⟨WORTE3⟩

Das Wort WHILE liest das oberste Element des Stapels und prüft, ob es

0 oder 1 ist, als Ergebnis eines Vergleichs, der unmittelbar vor dem WHILE ausgeführt wurde. Ist diese Bedingung erfüllt (f = 1) dann werden die WORTE2 zwischen WHILE und REPEAT ausgeführt. Danach geht das Programm zurück zum BEGIN und durchläuft WORTE1 bis zum WHILE. Wird durch einen Vergleich jetzt f = 0, so wird die Wortfolge WORTE3 nach dem REPEAT durchlaufen und damit die Schleife verlassen.

Die Worte zwischen WHILE und REPEAT werden sooft ausgeführt, solange die Bedingung vor dem WHILE erfüllt ist.

Beispiel:

Das folgende Programm in Abbildung 3.3 bildet ebenfalls, wie das Programm in Abbildung 3.2 die Summe der Zahlen von 1 bis 100.

```
: TEST 1 + DUP 101 < ;

: ADD DUP ROT + SWAP ;

: SUM   BEGIN TEST WHILE ADD REPEAT
        SWAP . ;


0 SUM 5050  OK
```

Abbildund 3.3
Bildung der Summe von 1 bis 100

Im Wort TEST wird das oberste Element des Stapels laufend um Eins erhöht und geprüft ob es kleiner als 101 ist.

Das Aufaddieren der Zahlen erfolgt in ADD. Die oberste Zahl auf dem Stapel, die Anzahl der bisher addierten Zahlen, wird verdoppelt und nach unten geschoben. Dann werden die beiden obersten Zahlen addiert. Auf dem Stapel steht nun die Summe, gefolgt von der Anzahl

der Zahlen. Vor dem Rücksprung nach BEGIN wird diese Reihenfolge wieder vertauscht sodaß die Anzahl im Teil TEST wieder um Eins erhöht werden kann.

Wird dabei die Zahl 101 erreicht, dann ist die Bedingung vor dem WHILE nicht mehr erfüllt und der Programmteil ADD wird übersprungen. Die Summe wird auf den Stapel geholt und ausgegeben.

Verlassen einer Schleife:

Das Wort LEAVE führt zum Abbruch einer Schleife.
Die Definition lautet:

LEAVE (n --> n)

Abbruch einer Schleife.

Das folgende Beispiel gibt die Zweierpotenzen, solange sie eine positive 16 bit Zahl sind, auf den Bildschirm aus.

```
: AUS CR  1 0 1000 0 DO + DUP DUP 0 >
      IF DUP 5 .R CR ELSE LEAVE THEN LOOP ;
```

```
 CR                    128
AUS                    256
    1                  512
    2                 1024
    4                 2048
    8                 4096
   16                 8192
   32                16384
   64                 OK
```

Für die obere Grenze der Schleife ist dabei die Zahl 1000 gewählt worden. Durch LEAVE wird die Schleife schon vorher verlassen.

3.6. Programmbeispiele

In diesem Kapitel sollen die bisherigen Definitionen in einfachen Programmbeispielen angewendet werden. Diese Programme sind bewußt sehr kurz und einfach gehalten, damit man sich in die Art und Weise, wie in FORTH programmiert wird, einarbeiten kann.

3.6.1. Berechnung des GGT zweier Zahlen

Für die Berechnung des größten gemeinsamen Teilers zweier Zahlen wird der Euklid'sche Algorithmus verwendet.

Zuerst bestimmt man den Rest R zweier ganzen Zahlen A und B durch (A MOD B). Ist R = 0, dann ist B der gesuchte GGT. Ist R ≠ 0, so wird A B und B = R. Danach wird die obige Modulo Division wieder ausgeführt, solange, bis die Bedingung R = 0 erfüllt ist.

Für diese Berechnung definieren wir ein Wort GGT.

```
: GGT BEGIN SWAP OVER MOD DUP 0= UNTIL
      DROP . ;
```

```
27 21 GGT 3  OK
```

Zum besseren Verständnis des Programmablaufs bei der Berechnung des GGT von 2 Zahlen, ist die Belegung des Stapels in Abbildung 3.4 gezeigt. Die Schleife zwischen BEGIN und UNTIL wird solange durchlaufen, bis bei der Modulodivision MOD kein Rest mehr auftritt. Die zweite Zahl auf dem Stapel ist dann der größte gemeinsame Teiler der beiden eingegebenen Zahlen.

STAPEL					TOS	EINGABE
					27	27
				27	21	21
				27	21	BEGIN
				21	27	SWAP
			21	27	21	OVER
				21	6	MOD
			21	6	6	DUP
			21	6	0	0=
				21	6	UNTIL
				6	21	SWAP
			6	21	6	OVER
				6	3	MOD
			6	3	3	DUP
			6	3	0	0=
				6	3	UNTIL
				3	6	SWAP
			3	6	3	OVER
				3	0	MOD
			3	0	0	DUP
			3	0	1	0=
				3	0	UNTIL
					3	DROP

Abbildung 3.4
Stapelbelegung beim Programmablauf GGT

3.6.2. Fibboconacci-Zahlen

Die Fiboconnacci-Zahlen stellen eine Zahlenfolge dar, bei welcher das nächste Element der Folge aus der Summe der beiden Vorgänger berechnet wird. Die ersten beiden Zahlen sind 0 und 1.

Die Folge lautet dann:

0 1 1 2 3 5 8 13 21 34 55

Das Programm dazu zeigt Abbildung 3.5.

```
( FIBBOCONACCI ZAHLEN )
: FIB ( ENDE --> ) CR  0 1 BEGIN
            DUP >R ROT DUP R> > WHILE
              ROT ROT DUP ROT + DUP .
                              REPEAT ;
```

Abbildung 3.5
Fibboconacci Zahlen

Vor dem Aufruf des Wortes FIB muß eine Zahl auf dem Stapel sein, die angibt, nach welcher Grenze die Entwicklung abgebrochen werden soll.

Für diese Programmverzweigung wird die BEGIN . . . WHILE . . . REPEAT Schleife verwendet. Die gerade berechnete neue Zahl, der Folge wird kurzzeitig auf dem RETURN-Stapel zwischengespeichert, um sie dann zum Vergleich wieder auf den Rechenstapel zurückzuholen.

Die ersten Durchläufe der Schleife wird in Abbildung 3.6 gezeigt.

Die Programmteile zwischen BEGIN und WHILE und zwischen WHILE und REPEAT werden solange durchlaufen, bis die Bedingung vor dem WHILE, Grenze größer als neue Zahl nicht mehr erfüllt ist. In unserem Beispiel ist dies bei der Bildung der achten Zahl der Folge (21) der Fall, und das Programm damit beendet.

Der Aufruf von 20 FIB ergibt:

20 FIB 1 2 3 5 8 13 21 OK

3.6.3. Berechnung der Primzahlen zwischen zwei Grenzen

Das folgende Programm berechnet die Primzahlen zwischen einer oberen und einer unteren Grenze. Vor dem Aufruf von PRIM müssen diese beiden Zahlen in dieser Reihenfolge auf dem Stapel stehen.

					20	20
					20	FIB
				20	0	0
			20	0	1	1
			20	0	1	BEGIN
		20	0	1	1	DUP
			20	0	1	⟩R
			0	1	20	ROT
		0	1	20	20	DUP
	0	1	20	20	1	R⟩
		0	1	20	1	⟩
			0	1	20	WHILE
			1	20	0	ROT
			20	0	1	ROT
		20	0	1	1	DUP
		20	1	1	0	ROT
			20	1	1	+
		20	1	1	1	DUP
			20	1	1	.
			20	1	1	REPEAT
		20	1	1	1	DUP
			20	1	1	⟩R
			1	1	20	ROT
		1	1	20	20	DUP
	1	1	20	20	1	R⟩
		1	1	20	1	⟩
			1	1	20	WHILE
			1	20	1	ROT
			20	1	1	ROT
		20	1	1	1	DUP
		20	1	1	1	ROT
			20	1	2	+
		20	1	2	2	DUP
			20	1	2	.
			20	1	2	REPEAT
		20	1	2	2	DUP

Abbildung 3.6
Stapelbelegung des Programmes FIB

```
( BERECHNUNG DER PRIMZAHLEN ZWISCHEN )
( ZWEI GRENZEN )

4 VARIABLE ZEILE

: TEST MOD 0= ;

: DRUCKE DUP 4 .R ZEILE @ DUP 0= IF CR DROP 4
         ELSE 1 - THEN  ZEILE ! ;

: PTEST DUP 2 / 2 DO DUP I TEST IF 0 LEAVE
        THEN  LOOP DUP IF DRUCKE ELSE DROP
        THEN  DROP ;

: PRIM ( ENDE ANF -->   )
       CR 4 ZEILE !
       DO I PTEST LOOP CR ;
```

```
200 1 PRIM
   1   3   5   7  11
  13  17  19  23  29
  31  37  41  43  47
  53  59  61  67  71
  73  79  83  89  97
 101 103 107 109 113
 127 131 137 139 149
 151 157 163 167 173
 179 181 191 193 197
 199
 OK
```

Abbildung 3.7
Primzahlen

Der verwendete Algorithmus ist sehr einfach. In einer Schleife PTEST wird probiert, ob die augenblicklich ausgewählte Zahl durch den Laufparameter der Schleife teilbar ist oder nicht. Die Schleife beginnt mit 2, die obere Grenze entspricht der Hälfte der zu untersuchenden Zahl.

Wird ein Teiler ohne Rest gefunden, so wird mit LEAVE die Schleife verlassen. Vor der Ausführung von LEAVE wird auf dem Stapel eine Null zurückgelassen. Dies beudetet aber für die folgende IF Abfrage die Bedingung FALSE, sodaß das Wort DRUCKE nicht ausgeführt, sondern das DROP nach dem ELSE ausgeführt wird.

Wird eine Zahl gefunden, die durch nichts teilbar ist, so wird diese im Wort DRUCKE auf den Bildschirm ausgegeben.

Das Wort DRUCKE führt auch eine Formatierung der Ausgabe durch. In einer Zeile werden jeweils 5 Zahlen ausgedruckt. Dazu wird eine Variable ZEILE benutzt, deren Anfangswert 4 ist.

Nach der Ausgabe einer Zahl wird der Wert dieser Variablen geholt und verglichen, ob dieser Wert 0 ist. Ist dies der Fall, so wird CR für eine neue Zeile ausgegeben und der Wert von ZEILE wieder auf 4 gesetzt. War der Wert nicht Null, so wird eins abgezogen und der Wert zurückgespeichert.

Das Programm ist in Abbildung 3.7 gezeigt.

3.6.4. Hexdump

Mit den Befehlen C@ und C!, die einzelne Byte aus dem Speicher lesen oder abspeichern, ist es leicht eine Hexadezimalausgabe auf den Bildschirm zu programmieren. In den meisten FORTH-Versionen ist dies unter dem Wort DUMP schon vorhanden.

HEX 1000 10 DUMP

listet den Inhalt der Zellen 1000 bis 1010 Hex auf den Bildschirm aus.

Hier wollen wir das Programm neu schreiben, wobei die Zahl hinter der Adresse die Zahl der Zeilen, die ausgegeben werden sollen angibt.

Das Programm zeigt Abbildung 3.8.

```
( HEXDUMP )    HEX

: ZEILE DUP DUP 8 + SWAP DO I C@ 3 .R LOOP ;

: NR CR CR 6 SPACES 8 0 DO I 3 .R LOOP CR ;

: A0 DROP 20 ;

: ATARI DUP 7D = IF A0 THEN DUP FD = IF A0 THEN
        DUP 1C = IF A0 THEN DUP 7E = IF A0 THEN
        DUP 1D = IF A0 THEN DUP 9C = IF A0 THEN
        DUP 1E = IF A0 THEN DUP FE = IF A0 THEN
        DUP 1F = IF A0 THEN
        DUP FF = IF A0 THEN
        DUP 90 = IF A0 THEN ;

: ASCII DUP DUP 8 + SWAP DO I C@ ATARI EMIT
        2 SPACES LOOP ;

: HDUMP CR DUP . SPACE ZEILE CR 8 SPACES
        ASCII 8 + ;

: DUMP ( ADR N --> )
       NR 0 DO HDUMP LOOP ;
                                DECIMAL

HEX  OK
1000 5 DUMP

        0  1  2  3  4  5  6  7

1000   85 53 54 41 54 C5 F2  F
          S  T  A  T  E  r
1008   5E  8  0  0 83 42 4C CB
        ^           B  L  K
1010    0 1C 5E  8  0  0 83 3E
           ^              >
1018   49 CE  C 1C 5E  8  B  0
        I  N

                   ^
1020   A4 4D 4F 44 C5 16 1C 5E
        $  M  O  D  E        ^    OK
```

Abbildung 3.8
Hexdump

Es wird in die Zahlendarstellung HEX umgeschaltet. Dann wird mit ZEILE die Ausgabe einer Zeile definiert. Die Anfangsadresse wird dupliziert und nachdem 8 dazuaddiert wurde, werden die beiden Zahlen vertauscht. Sie sind die obere und untere Grenze der folgen den Schleife. Der Inhalt der einzelnen Zellen (I ist Laufparameter der Schleife und somit gleichzeitig die Adresse der Speicherzelle) wird geholt und in einem 3 Zeichen breiten Zahlenfeld rechtsbundig ausgegeben.

Mit NR wird eine Überschrift 0 bis 7 über der ersten Zeile ausgegeben.

Der Inhalt der Speicherzellen soll gleichzeitig, eine Zeile tiefer als ASCII-Zeichen ausgegeben werden.
Dieses Programm wurde auf einem ATARI 800 entwickelt. Dieser Rechner verwendet einige Zeichen zur Cursorsteuerung und zum Löschen des Bildschirms oder von Zeilen. Das Zeichen 7D löscht, wenn es mit EMIT ausgegeben wird, den Bildschirm.

Dieses und die anderen Zeichen werden unter dem Wort ATARI durch Leerzeichen ersetzt. Das vorher definierte Wort A0 verringert die Schreibarbeit. Dieses Umsetzen der Zeichen wird durch einzelne IF . . . ENDIF Abfragen erledigt. Dabei wird allerdings, auch wenn z. B. das Zeichen 7B gleich beim ersten Vergleich gefunden wird, alle anderen Vergleiche trotzdem ausgeführt.

Eine bessere und schnellere Möglichkeit wäre es, die Abfrage durch geschachtelte IF . . . ELSE . . . ENDIF Anweisungen zu programmieren. Eine solche Abfrage für 3 verschiedene Zeichen zeigt das folgende Beispiel:

```
: A0 DROP 20 ;

: ATARI DUP 7D = IF A0 ELSE
        DUP 1C = IF A0 ELSE
        DUP 1D = IF A0 ELSE
        THEN THEN THEN ;
```

Wird in diesem Fall das Zeichen 7D gefunden, so wird A0 ausgeführt und alles nach dem ELSE bis zum letzten THEN übergangen.

Allerdings sind es nur wenige Steuerzeichen, so daß diese Art der Programmierung keine wesentliche Beschleunigung der Programmlaufzeit mit sich bringt.

Mit ASCII wird die gleiche Zeile nunmehr als ASCII-zeichen ausgegeben.

In HDUMP wird die Adresse ausgegeben, danach Zeile und ASCII. Mit H+ wird die Anfangsadresse der nächsten Zeile berechnet.

Dies alles ist dann in dem Wort DUMP zusammengefasst.

Die Eingabe HEX 1000 8 DUMP druckt den Inhalt der Speicherzellen 1000 HEX bis 103F HEX aus.

3.7. Zusammenstellung der Definitionen

DO . . . LOOP (n1 n2 n3 -- > n1)

Schleife wird von n3 bis n2 in Einerschritten durchlaufen. n2 > n3

I (n1 n2 n3 -- > n1 n2 n3 n)

Der Laufparameter n der Schleife wird auf den Stapel gelegt.

+LOOP (n1 n2 n3 -- > n1 n2)

Der Zählparameter der Schleife wird um n3 verändert.

< (n1 n2 n3 -- > n1 f)

f = 1 wenn n2 < n3

> (n1 n2 n3 -- > n1 f)

f = 1 wenn n2 > n3

= (n1 n2 n3 -- > n1 f)

f = 1 wenn n2 = n3

0< (n1 n2 n3 -- > n1 n2 f)

f = 1 wenn n3 < 0

0= (n1 n2 n3 -- > n1 n2 f)

f = 1 wenn n3 = 0

IF <WORTE> ENDIF (THEN)
IF (n1 n2 f -- > n1 n2)

Die <WORTE> zwischen IF und ENDIF werden nur ausgeführt, wenn f = 1 ist.

IF <WORTE1> ELSE <WORTE2> ENDIF (THEN)

IF (n1 n2 f -- > n1 n2)

Die <WORTE1> werden ausgeführt, wenn f = 1 ist. Mit f = 0 werden die <WORTE2> ausgeführt.

BEGIN <WORTE> <BEDINGUNG> UNTIL

UNTIL (n1 n2 f -- > n1 n2)

Die Ausführung der <WORTE> wird solange wiederholt, bis die <BEDINGUNG> erfüllt, f = 1 ist.

BEGIN <WORTE1> <BEDINGUNG> WHILE <WORTE2> REPEAT <WORTE3>

WHILE (n1 n2 f -- > n1 n2)

Solange die Bedingung, f = 1, erfüllt ist, wird <WORTE1> und <WORTE2> durchlaufen. Mit f = 0 wird <WORTE2> übersprungen und <WORTE3> ausgeführt.

KEY (n1 n2 n3 --> n1 n2 n3 c)

Warten auf Tasteneingabe.

NOTIZEN

4 Zahlensysteme

4. Zahlensysteme

In den bisherigen Beispielen wurde ausschließlich das 16 bit ganzzahlige Zahlensystem verwendet. Obwohl hier der Zahlenbereich nur von 32768 bis +32767 reicht, ist er doch für die meisten Anwendungen in Steuerungs- und Regelsystemen ausreichend. Die für die Umwandlung von analogen Spannungen in digitale Zahlen verwendeten Wandler haben selten mehr als 12 bit Auflösung. Das entspricht einem Zahlenbereich von –2048 bis +2047.

Müssen diese Zahlenwerte umgerechnet werden, so ist es in den meisten Fällen nicht notwendig auf Fließkommarechnung umzusteigen, sondern man wird eine Skalierung der Werte vornehmen. Dies ist eine Verfahrensweise, wie sie oft in der Analogrechentechnik angewendet wird.

Der Vorteil dieser Umrechnung liegt darin, daß man weiter die schnellen 16 bit Rechenoperationen verwenden kann und nicht die langsamen Fließkommaroutinen.

4.1. Skalierung von Rechenabläufen

Das Wort Skalierung bezeichnet die Verwendung eines anderen Maßstabs (scale) bei der Umrechnung von Zahlenwerten.

Betrachten wir folgendes Beispiel:
Werden in einem Programm Geldbeträge angegeben, so wird als Einheit für diese Angabe 1 DM (eine Deutsche Mark) verwendet. Die Angabe von 1.25 bedeutet eine Mark + 25/100 Mark. Die Zahl 1.25 ist eine Gleitkommazahl.

Diese Gleitkommazahl können wir durch Wahl eines anderen Maßstabs, einer anderen Einheit in eine Festkommazahl umwandeln. Verwenden wir anstatt der Einheit Mark die Einheit Pfennig für die Angabe eines Geldbetrages, so haben wir statt 1.25 DM 125 Pf, und somit eine Festkommazahl.

Ein zweites Beispiel:
Mit einem 8-bit Analog-Digital Wandler wird eine Temperatur in einem Bereich zwischen 0° Celsius und 100° Celsius gemessen. Die Auflösung dieses Temperaturbereichs mit einem 8-bit Wandler beträgt 100/255 = 0.3921 °C

Für die Umrechnung der gemessenen 8 bit Zahl in die entsprechende Temperatur verwenden wir die Zahl 392 als Einheit.
In FORTH gibt es für diese Skalierungsberechnung einen eigenen Operator */. Die Definition dieses Operators ist

```
*/          ( n1 n2 n3 --> n )

n = n1 * n2 / n3
```

Das Besondere dieses Operators ist aber, daß das Ergebnis der Multiplikation als doppelt lange Zahl zwischengespeichert wird. Diese doppelt lange Zahl wird durch n3 geteilt.

Für die Umrechnung der Messwerte in den Temperaturwert definieren wir uns ein Wort SCALE.

```
: SCALE 392 100 */ . ;  OK
```

Damit errechnet sich:

```
233 SCALE  913  OK

232 SCALE 909  OK

231 SCALE 905  OK
```

Die Zahl 913 ist als 91.3 zu interpretieren, da wir mit 392 multiplizieren, aber nur durch 100 teilen.

Der Messwert 232 = 90.9 °C bedeutet aber nur, daß die Temperatur zwischen 90.5° und 91.3 °C liegt. Der angegebene Zahlenwert kann nie genauer sein, als die Auflösung des Wandlers.

Man kann die Zahlenwerte noch etwas genauer machen, wenn man den verbleibenden Rest bei der Division durch 100 bedrachtet und die letzte Stelle des Ergebnisses entsprechend aufrundet. Dazu kann man den Operator */MOD verwenden.

```
*/MOD   ( n1 n2 n3 --> n n4 )
```

n4 = n1 * n2 / n3 n = Rest der Division.

Mit einer neuen Definition von SCALE

```
: SCALE 392 100 */MOD SWAP 50 + 100 / + . ;   OK
```

erhalten wir `231 SCALE 906  OK`

Da dieser Zahlenwert aber auch nur auf ± 0.4 % genau ist, wird es für die meisten Anwendungen genügen, die Temperatur in ein Grad Schritten anzugeben. Dazu das folgende Programm in Abbildung 4.1

```
( MESSWERT-UMRECHNUNG )

: ROUND SWAP 500 + 1000   + ;

: SCALE 392 1000 */MOD ROUND ;

: M->T SCALE . CR ;

233 M->T 91
```

Abbildung 4.1
Messwert-Umrechnung

Mit dem Taschenrechner hätte man den Messwert 233 folgendermaßen umgerechnet:

$$T = \frac{233}{255} * 100 = 91.372549$$

In vielen Fällen wird ein Messergebnis in dieser Form angegeben. Dabei wird aber nicht beachtet, daß die Zahlen nach dem Dezimalpunkt, mit

Ausnahme der ersten Stelle, reine Zufallszahlen sind und in keinem Zusammenhang mit dem Messwert haben.

4.2. Doppelt lange Zahlen

Falls die Genauigkeit von 16 bit Zahlen nicht ausreicht, wird mit doppelt langen 32 bit Zahlen gerechnet. Der Zahlenbereich reicht dann von – 1.073.741.824 bis + 1.073.741.823.
Eine doppelt lange Zahl wird mit einem Dezimalpunkt eingegeben. An welcher Stelle dies geschieht ist völlig gleichgültig.

123.4567
1.234567
1234567.
1.23.567

sind völlig gleiche doppelt lange Zahlen. In poly-FORTH kann ein beliebiges numerisches Zeichen verwendet werden wie z. B.

6/02/82

Dies wird auch als doppelt lange Zahl in den Stapel übernommen.

Für die Verarbeitung von doppelt langen Zahlen sind folgende Worte vereinbart:

D. (d1 d2 d3 --> d1 d2)

Ausgabe von d3

D+ (d1 d2 d3 --> d1 d)

d ist die Summe von d1 und d2

D– (d1 d2 d3 --> d1 d)

d ist die Differenz von d2 und d1

f 1, wenn d2 = d3

D0= (d1 d2 d3 --> d1 d2 f)

f = 1 wenn d = 0

D< (d1 d2 d3 --> d1 f)

f = 1 wenn d3 < d2

DMAX (d1 d2 d3 --> d1 d)

d = d2 wenn d2 > d3 und d = d3 wenn d3 > d2 ist.

DMIN (d1 d2 d3 --> d1 d)

d = d2 wenn d2 < d3 und d = d3 wenn D3 < d2 ist.

D.R (d1 d2 d3 --> d1)

Druckt d2 in einen d3 breiten Zahlenfeld aus.

Beispiele:

```
10000000. 1. D+ D.  10000001 OK
```

oder

```
1000000020. 1. D+ D. 1000000021 OK
```

Dies ist eine Addition, bei welcher die meisten Basic Interpreter keine genauen Rechenergebnisse mehr liefern.

Das bedeutet aber, daß man mit FORTH Geldbeträge bis zu 10 Millionen DM auf den Pfennig genau berechnen kann.

Weitere Beispiele brauchen hier nicht gemacht zu werden, da die Be fehle wie bei einfach langen Zahlen ablaufen.

4.3. Gemischte Zahlenoperatoren

Bei den Rechenoperationen für doppelt lange Zahlen fehlen die Multi plikation und die Division. Eine Multiplikation zweier 32 bit Zahlen würde ein 64 bit Ergebniss liefern. Zwei Operatoren erlauben aber das Arbeiten mit gemischten, also 16 bit und 32 bit Zahlen. Das Wort M* bildet das 32 bit Produkt zweier 16 bit Zahlen.

M* (n1 n2 --> d1)

d1 32 bit Produkt von n2 * n1.

Beispiel:

```
100 400 M* D. 40000 OK
```

Das Wort M/ dividiert eine 32 bit Zahl durch eine 16 bit Zahl, das Ergebnis ist ebenfalls eine 16 bit Zahl.

Beispiel:

```
40000. 20 M/ . 2000  OK
```

Diese gemischten Zahlenoperatoren wird man immer dann verwenden, wenn bei einer Multiplikation das Produkt größer als 32767 wird.

Einen anderen gemischten Operator haben wir früher schon verwendet, nämlich den Scalierungsoperator */ . Die Definition unter Verwendung von M* und M/ wäre dann:

```
: */ M* M/ ;
```

4.4 Zahlen ohne Vorzeichen

Eine 16 bit Zahl auf dem Stapel wird normalerweise als eine 15 bit Zahl plus Vorzeichen behandelt. Das Konvertierungswort NUMBER das FORTH intern beim Compilieren oder beim Interpreter verwendet, überprüft bei der Eingabe nicht, ob der Zahlenbereich überschritten wird oder nicht. Bei der Eingabe der folgenden Zahlen erhalten wir als Ausgabe:

```
65535 .  -1  OK
65536 . 0  OK
65537 . 1  OK
```

Wird also eine Zahl größer 65535 eingegeben, so wird sie als n – 65536 gewandelt.
Diese Zahlen kann man aber auch als Zahlen ohne Vorzeichen betrachten.

Mit 65535 U. erhält man wieder die Zahl 65535 als Ausgabe.

U. (n1 u --> n1)

Ausgabe der Zahl im TOS als Zahl der Vorzeichen (0 bis 65535).

Bei einigen Rechnern werden Adressen, die größer als 32767 sind als negative Dezimalzahlen angegeben. Die Umrechnung einer solchen Dezimaladresse in die Hexadezimaladresse zeigt das folgende Beispiel:

```
-151 HEX U. FF69  OK
```

Hätten wir nur die normale Ausgabe verwendet, so hätten wir

```
-151 HEX . -97  OK
```

erhalten.

Folgende Operationen sind mit Zahlen ohne Vorzeichen möglich:

U* (u1 u2 -- ud)

Zwei 16 bit Zahlen ohne Vorzeichen u1 und u2 werden multipliziert. Das Ergebnis ist ein 32 bit Produkt ebenfalls ohne Vorzeichen.

Beispiel:

```
32767 DUP U* 0 D.R 1073676289 OK
```

U/MOD (ud u1 -- u2 u3)

Dividiert eine 32 bit durch eine 16 bit Zahl, beide ohne Vorzeichen. Das Ergebnis ist der Quotient und der Rest. Beides 16 bit Zahlen ohne Vorzeichen.

U< (u1 u2 -- f)

f = 1 wenn u1 < u2 ist, beides 16 bit Zahlen ohne Vorzeichen.

Beispiel:

```
1 65535 U< . 1  OK
```

aber:

```
1 65535 < . 0  OK
```

In FORTH 79 ist auch eine Schleifenbildung mit Zahlen ohne Vorzeichen definiert.

Do (u1 u2 --)

u1 obere Grenze, u2 untere Grenze.

/LOOP (u -->)

u Schrittweite.

Wie bei der DO . . . +LOOP Schleife muß der Zählparameter von /LOOP angegeben sein.

Beispiel:

```
: T 60000 0 DO CR I U. 4098 /LOOP ;

T
0
4098
8196
12294
16392
20490
24588
28686
32784
36882
40980
45078
49176
53274
57372  OK
```

4.5. Formatierung von Zahlenausgaben

In den meisten Fällen werden Zahlen formatiert ausgegeben. Ein Preis z. B. als 2486.23 DM die Uhrzeit als 10:21:32 oder eine Telefonnummer als 0731–817536. FORTH stellt für solche Fälle Worte zur Formatierung zur Verfügung.

Beispiel:
Wir definieren ein Wort uD. das uns eine 32 bit Zahl als vorzeichenlose Zahl ausgibt. Die Definition lautet:

```
: UD. <# #S #> TYPE SPACE ;
```

Dabei bedeutet:

<# (ud --> ud)

Beginn der Zahlenformatierung.

#S (-->)

Alle Stellen der Zahl werden zu ASCII-Zeichen gewandelt.

#> (--> adr n)

Ende der Formatierung. Auf dem Stapel bleibt die Adresse der ASCII-Zeichenkette und die Anzahl der Zeichen.

TYPE (adr n -->)

Ausgabe einer Zeichenkette. Auf dem Stapel muß die Adresse dieser Zeichenkette und die Zahl der Zeichen sein (siehe Kapitel 7).

SPACE (n --> n)

Ausgabe eines Leerzeichens.

Beispiel:

```
65535 DUP U* UD. 4294836225  OK
```

Bei dieser Umsetzung einer Zahl in eine Zeichenkette können nun auch andere ASCII-Zeichen eingefügt werden. Wir wollen ein Wort .DM definieren, das uns eine Zahl (doppelt-lang) mit einem Dezimalpunkt vor den letzten beiden Stellen ausgibt. Also 65358 als 653.58 . Unsere Recheneinheit für Geldbeträge sei 1 Pfennig.

```
: .DM <# # # 46 HOLD #S #>
      TYPE SPACE ;
```

Beispiel: `65358. .DM 653.58  OK`

In dieser Definition sind 2 neue Worte hinzu gekommen.

(-->)

Umwandlung einer Stelle.

HOLD (c -->)

Einfügen des ASCII-Zeichen c in die Zeichenkette.

In der Definition .DM wird die Umsetzung der Zahl mit <# begonnen. Dann werden die beiden letzten Stellen gewandelt. Die Umwandlung beginnt also bei den letzten beiden Stellen. Nun wird mit 46 HOLD das ASCII-Zeichen für einen Dezimalpunkt in die Zeichenkette eingeführt. Das Wort #S wandelt nun die restlichen Stellen um und #> beschließt die Formatierung.

In dem Beispiel wurde die Zahl 65358 als doppeltlange Zahl (65358.) eingegeben. Liegt nur eine einfach lange Zahl vor, die formatiert werden soll, so muß vor der Formatierung eine Null auf den Stapel abgelegt werden. Die Eingabe

```
65358 0 .DM
```

bringt ebenfalls die Ausgabe

```
653.58  OK
```

Das folgende kleine Programm rechnet Sekunden in Stunden, Minuten und Sekunden um (aus L. Brodie, FORTH Inc., Starting FORTH).

```
: SEXTAL 6 BASE ! ;
: :00 # SEXTAL # DECIMAL 58 HOLD ;
: SEC <# :00 :00 #S #> TYPE SPACE ;
```

Zuerst wird ein Wort SEXTAL definiert, das eine Rechnung in einem

6er Zahlensystem ermöglicht. Denn bei der Wandlung in Sekunden oder in Minuten kann die letzte Stelle die Zahlen Null bis Neun annehmen, die nächste Stelle aber nur die Zahlen Null bis Fünf. Dies wird in der Definition :00 berücksichtigt. Zuerst wird in Dezimalrechnung eine Stelle gewandelt, dann auf Rechnung im 6er System umgeschaltet und die nächste Stelle gewandelt. Danach wird auf Dezimalrechnung zurückgeschaltet und mit 58 HOLD ein Doppelpunkt ausgegeben. Das Wort SEC wandelt zuerst die Sekunden, dann die Minuten und den verbleibenden Rest in Stunden um.

```
3601. SEC 1:00:01  OK
```

Auf ähnliche Weise lassen sich auch andere Zahlen formatieren und ausgeben.

4.6. Gleitkommazahlen

Bei den bisherigen Berechnungen haben wir nur einfach lange (16 bit) und doppelt lange (32 bit) Zahlen verwendet. Natürlich gibt es auch FORTH Worte zum Rechnen in Gleitkommadarstellung. Aber es liegt der Philosophie der Sprache zugrunde, daß das Rechnen damit vermieden wird.

Am Anfang dieses Kapitels haben wir auf das Skalieren und Rechnen in Einheiten schon hingewiesen. Bei vielen Anwendungen sind die Einheiten durch die Auflösung der Wandler gegeben. Bei einem 8 bit Wandler sind die gewandelten Zahlen ganze Zahlen im Bereich von 0 bis 255 und nicht 0.00 bis 255.00 . Bei kaufmännischen Berechnungen wird die Einheit immer ein Pfennig sein. Wir haben gesehen, daß durch die Verwendung von doppelt langen Zahlen, man Geldbeträge bis 10 Millionen DM in Pfennigeinheiten ausdrücken kann.

Es ist also offensichtlich, daß man sehr oft auf die Rechnung in Gleitkommadarstellung verzichten kann. Gleichzeitig verzichtet man dabei auf die auftretenden Rundungsfehler.

In den wenigen Fällen, wo man auf eine Berechnung in Gleitkomma-

Darstellung nicht verzichten kann, wird man ein Gleitkommapaket, das zu den meisten FORTH-Versionen erhältlich ist, hinzuladen.

Auf eine Darstellung der definierten Worte sei hier verzichtet, da diese nicht standartmässig vereinbart sind. Sie können aus den jeweiligen Beschreibungen entnommen werden.

4.7. Zusammenstellung der Definitionen

*/ (n1 n2 n3 -- > n)

n = n1 * n2 / n3. Das Produkt n1 x n2 wird doppelt lang berechnet.

*/MOD (n1 n2 n3 -- > n4 n)

n = n1 * n2 / n3 Rest der Division ist n4

D. (d1 d2 d3 -- > d1 d2)

Ausgabe von d3

D+ (d1 d2 d3 -- > d1 d)

d ist die Summe von d1 und d2

D- (d1 d2 d3 -- > d1 d)

d ist die Differenz von d2 und d1

D= (d1 d2 d3 -- > d1 f)

f = 1, wenn d2 = d3

D0= (d1 d2 d3 --> d1 d2 f)

f = 1 wenn d = 0

D< (d1 d2 d3 --> d1 f)

f = 1 wenn d3 < d2

DMAX (d1 d2 d3 --> d1 d)

d = d2 wenn d2 > d3 und d = d3 wenn d3 > d2 ist.

DMIN (d1 d2 d3 --> d1 d)

d = d2 wenn d2 < d3 und d = d3 wenn d3 < d2 ist.

D.R (d1 d2 d3 --> d1)

druckt d2 in einen d3 breiten Zahlenfeld aus.

M* (n1 n2 --> d1)

d1 = n1 * n2 32 bit Produkt.

M/ (d1 n1 --> n2)

n2 = d1 / n1

U. (n1 u --> n1)

Ausgabe von u als Zahl ohne Vorzeichen.

U/MOD (ud u1 --> u2 u3)

Dividiert eine 32 bit Zahl durch 16 bit Zahl. u3 ist Quotient, u2 der Rest der Division. Alles Zahlen ohne Vorzeichen.

(u1 u2 --> f)

vergleicht die beiden obersten Zahlen in Stapel als Zahlen ohne Vorzeichen.

DO /LOOP

DO (u1 u2 -->)

u1 obere Grenze, u2 untere Grenze.

LOOP (u -->)

u Schrittweite.

<# (ud --> ud)

Beginn der Zahlenformatierung.

#S (-->)

Alle verbleibenden Stellen werden in ASCII-Zeichen gewandelt.

(-->)

Wandlung einer Zahlenstelle

HOLD (c -->)

Einfügen des ASCII-Zeichens c in die Zeichenkette.

#> (--> adr n)

Ende der Formaterung. Auf dem Stepel bleibt die Adresse der Zeichenkette und die Zahl der Zeichen.

TYPE (adr n -->)

Ausgabe einer Zeichenkette, deren Anfang bei adr gespeichert ist und die n Zeichen lang ist (siehe Kapitel 7).

SPACE (-->)

Ausgabe eines Leerzeichens (siehe Kapitel 7).

5 Textfeld (SCREEN) und Editor

5. Textfeld (SCREEN) und Editor

Bei den bisherigen Beispielen wurde zum Teil schon für die Eingabe eines Programms sog. SCREEN's verwendet. Als deutsche Übersetzung dieses Wortes wird das Wort Textfeld verwendet. Für die Programmentwicklung ist die Eingabe solcher Textfelder unbedingt notwendig. Betrachten wir dazu nochmals das folgende Beispiel:

Auf dem Wörterbuchstapel befinden sich die Worte:

```
.S
TIEF
'S
SAVE                                    usw.
```

Hier wird das Wort 'S zum Beispiel in TIEF und in .S verwendet. Ist nun bei der Definition dieses Wortes ein Fehler gemacht worden, so genügt es nicht, 'S neu zu definieren, sondern man muß 'S vergessen und somit auch TIEF und .S . Danach können diese Worte neu definiert werden. Dies kann bei langen Programmen sehr umständlich sein. Deswegen wird ein Programm in eine Screen, in ein Textfeld mittels eines Editors geschrieben und diese dann durch einen Befehl compiliert.

5.1. Textfeld (Screen)

In der Abbildung 5.1 ist ein Textfeld abgebildet.

Dieses Textfeld besteht aus 16 Zeilen zu je 64 Zeichen. Es belegt also 1k Byte Speicherplatz. Diese Festlegung ist nicht überall gleich. Die Textfelder verschiedener FORTH-Versionen können davon abweichen.

763 LIST

```
 0 ( CLASS UTILITIES 2 EF 08 MAR 82 ) EMPTY
 1 : ?STOP 100 0 DO DUP I = IF LEAVE THEN I . LOOP DROP ;
 2 : TIMES 11 1 DO DUP I * . LOOP DROP ;
 3 : TIMES-TABLE CR 13 1 DO 13 1 DO I J * . LOOP CR LOOP ;
 4 : WINDOW CR 1 + SWAP DO 2DUP 1+ SWAP DO I J * . LOOP CR LOOP
 5     2DROP ;
 6 : GREED CR 1 + 1 DO I . ." FOR YOU " 5 SPACES I 1 + 1 DO I .
 7       LOOP ." FOR ME " CR LOOP ;
 8 : STARS 0 DO ." *" LOOP CR ;
 9 : BOX CR 0 SWAP 0 DO 2DUP DO ." * " LOOP CR LOOP 2DROP ;
10 : \STARS  0 DO  I SPACES 10 0 DO ." * " LOOP CR   LOOP ;
11 : /STARS   1 - 0 SWAP  DO I SPACES 10 0 DO ." * " LOOP CR -1
12        +LOOP ;
13 : XSTARS CR ROT ROT DO DUP /STARS DUP  \STARS LOOP DROP ;
14 : ZIP CR BEGIN DUP SPACES 10 0 DO ." * " LOOP 1 CR - DUP
15       NOT END DROP ;
```

Abbildung 5.1
Textfeld

Jedes Textfeld ist durch eine Zahl gekennzeichnet. Mit 763 LIST wurde dieses Textfeld auf den Drucker ausgegeben. Im Normalfall wird die Ausgabe auf den Bildschirm erfolgen.

Die Zeilen sind von 0 bis 15 durchnummeriert. Die erste Zeile enthält Kommentar. Der Kommentar beginnt mit einem (Zeichen gefolgt von einem Leerzeichen und geht bis zum Zeichen) . In diesen Kommentar sollte die Aufgabe, der Programmierer und das Datum der Erstellung dieses Textfeldes enthalten sein. In diesem Beispiel handelt es sich um ein Textfeld, in das Aufgaben während eines Seminars bei FORTH Inc. eingetragen wurden. In den folgenden Zeilen sind dann die Definitionen aufgeführt. Eine neue Definition sollte immer am Anfang einer Zeile beginnen (siehe auch Anhang C).

Ist mit Hilfe eines Editors ein Textfeld erstellt worden, so wird es mit dem Wort FLUSH auf Diskette geschrieben. Mit 763 LOAD wird dieses Textfeld compiliert.

Ein Textfeld ist auch gleichzeitig ein Datenfeld auf einem externen Speicher, in den meisten Fällen eine Diskette. Die Abbildung 5.2 zeigt den Zusammenhang zwischen Datenfeld auf Diskette und Datenfeld im Rechner. Wird mit 763 LIST ein Textfeld aufgerufen, so wird vom Betriebssystem nachgesehen ob dieses Datenfeld schon im Speicher

[illegible] ist. Wenn nicht, wird es in einen freien Block gelesen und [illegible] auf den Bildschirm ausgegeben. Dabei wird in den Kopf dieses [illegible]feldes im Speicher die Nummer des Textfeldes und ein UPDATE [illegible] geschrieben. Dieses Bit wird auf Eins gesetzt, wenn in dem Textfeld [illegible] Speicher eine Änderung durchgeführt worden ist. Das Wort [illegible]DATE kann auch als Befehl ausgeführt werden.

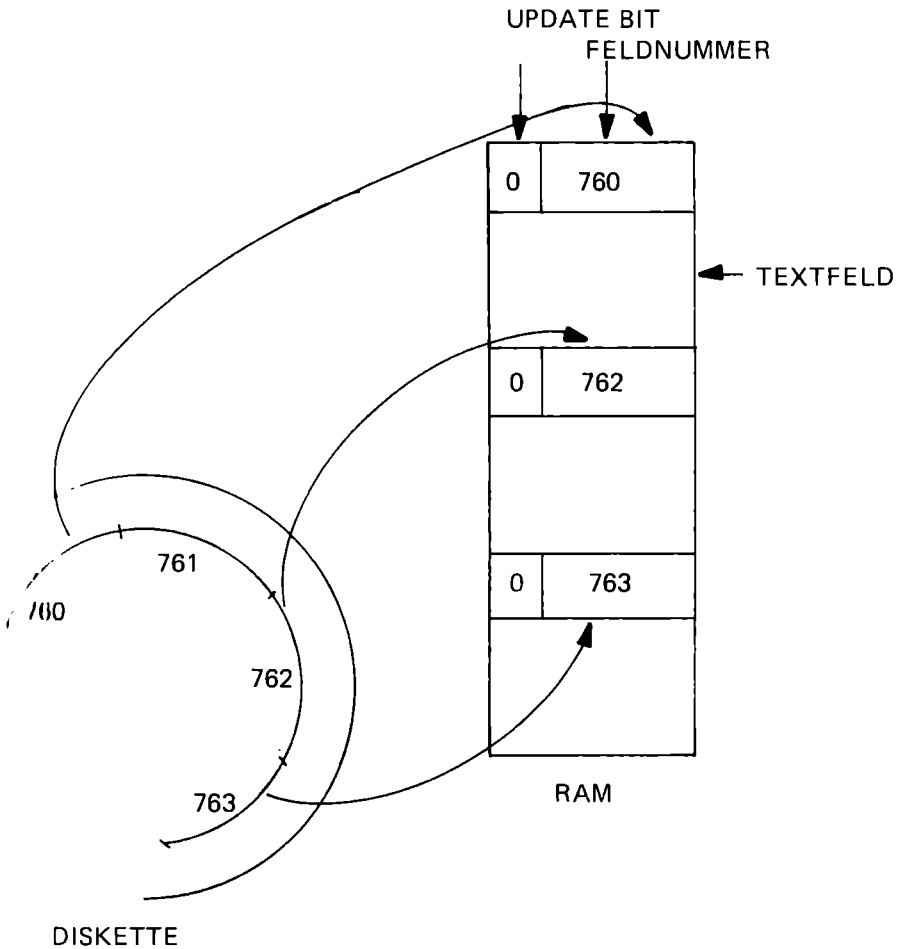


Abbildung 5.2
Textfelder auf Diskette und im RAM

Die Zahl der zur Verfügung stehenden Datenfelder ist natürlich durch die Speicherkapazität der Diskette gegeben. Auf einer Diskette mit 80k Byte Speicherplatz können also nur 80 Datenfelder gespeichert werden.

Ebenso wie bei LIST wird auch bei LOAD ein Datenfeld von der Diskette geholt, wenn es noch nicht im Speicher vorhanden ist. Bei der folgenden Compilierung werden die Definitionen ins Wörterbuch eingetragen.

Ist dieses Textfeld aber vorher schon einmal übersetzt worden, so wird bei der zweiten Compilierung laufend die Meldung ‹NAME› NOT UNIQUE ausgegeben werden. Um dies zu vermeiden, und um gleichzeitig zu verhindern, daß der Wörterbuchstapel mit fehlerhaften Definitionen anwächst, wird man vor Beginn der Definitionen eine leere Definition, z. B.

: TASK ;

einführen.

Nach der ersten Compilierung kann man mit

FORGET TASK

alle bisherigen Eintragungen löschen. Dieses FORGET TASK kann aber erst nach der ersten Compilierung in das Textfeld eingetragen werden, da sonst die Fehlermeldung TASK ? erscheint.

In Poly-FORTH ist das Wort EMPTY definiert. Dieses löscht das vom Benutzer aufgestellte Wörterbuch und kann somit immer gleich in der ersten Zeile nach dem Kommentar aufgeführt werden.

Reicht für ein Programm ein Textfeld nicht aus, so kann man mit dem Wort – – › das folgende Textfeld an das bisherige anhängen. Die Compilierung wird dort fortgesetzt. Eine Beendigung der Compilierung erfolgt entweder am Ende eines Textfeldes oder bei dem Wort ;S .

Soll der Inhalt eines Textfeldes gelöscht werden, so geschieht dies mit dem Wort CLEAR.

763 CLEAR

löscht den Inhalt des Textfeldes 763. Weiterhin kann man mit COPY Textfelder kopieren. Mit

763 765 COPY

wird das Textfeld 763 nach 765 übertragen.

9.7 Der Editor

Zum Erstellen eines Textfeldes wird ein Editor benötigt. Bei den auf verschiedenen Rechnern implementierten FORTH-Versionen werden unterschiedliche Editoren verwendet. Diese sind entweder bildschirmorientiert (QS-FORTH ATARI, FORTH-79 APPLE) oder Zeilenorientiert (poly-FORTH, APEX-FORTH ATARI) um nur zwei Beispiele zu nennen. Deshalb soll auf eine ausführliche Beschreibung des Editors verzichtet werden und nur kurz auf den zeilenorientierten Editor von poly FORTH eingegangen werden.

Mit

763 LIST

wird das Textfeld 763 auf den Bildschirm gebracht. Damit kann es nun editiert werden. Durch Eingabe von L (ohne Nummer) wird der Editor aufgerufen und das durch LIST aufgerufene Textfeld auf dem Bildschirm angezeigt. Soll nun Text in eine Zeile eingeführt werden, so wird mit 5 T z. B. die Zeile 5 angewählt. Der Inhalt dieser Zeile wird unten am Bildschirm ausgegeben.

Am Ende dieser Ausgabe wird angezeigt, um welche Zeile es sich handelt.

Ist in der Zeile kein Text vorhanden, so kann mit P (Put) Text in diese Teile eingegeben werden.

P EINGABE VON TEXT ⟨CR⟩ OK

Wenn man nun mit L dieses Textfeld wieder auf den Bildschirm ausgibt, so muß dort

erscheinen.

Bei der Eingabe zeigt ein Cursor ↑ auf die Stelle an welcher ein Buchstabe eingegeben wird. Wenn wir nun mit 5 T nochmals die Zeile 5 aufrufen und mit

P NEU ⟨CR⟩

eingeben, so ist der alte Text durch das Wort NEU ersetzt worden.

Um nun Buchstaben in einen Text einzusetzen oder zu löschen, muß man den Cursor des Editors an die passende Stelle setzen. Als Beispiel enthalte die Zeile 5 den fehlerhaften Text, der korrigiert werden soll:

HIER IST EIN FEELER

Mit 5 T wird die Zeile in den Eingabespeicher geholt und mit F (FIND) die Zeichenkette FEL gesucht.

F FEEL

setzt den Cursor rechts neben die Zeichenkette FEL. Mit

I H ⟨CR⟩

wird nun ein H nach FEE eingefügt und die korrigierte Zeile ausgegeben.

Soll ein Buchstabe gelöscht werden, so muß er zuerst gesucht werden. In unserem Text, der nach der letzten Korrektur so aussieht:

HIER IST EIN FEEHLER

soll das zweite E gelöscht werden. Mit

F E ⟨CR⟩

wird das erste E in HIER gefunden. Bei der Eingabe von F ist aber das

[illegible] einen Zwischenspeicher übernommen worden und ist dort noch [illegible]. Die Eingabe eines weiteren F findes das E in EIN und nach [illegible] weiteren F steht der Cursor hinter dem zusätzlichen E das mit [illegible] (BASE) gelöscht werden kann. Das Löschen einer Zeichenkette [illegible] mit D.

D FEE

[illegible] die Zeichenkette FEE in dem Wort FEEHLER mit der Wortfolge

5 T ⟨CR⟩
F N ⟨CR⟩
I FE ⟨CR⟩

kann nun das Wort korrigiert werden.

T 5 holt Zeile 5 und setzten den Cursor auf den Anfang. F N sucht die Zeichenkette N und setzt durch I FE danach FE in den Text ein.

Das Ganze hätte man aber wesentlich einfacher mit dem R (Replace) Befehl machen können. Mit

5 T ⟨CR⟩
F FEEHLER ⟨CR⟩
R FEHLER ⟨CR⟩

wird das folgende Wort auf einmal korrigiert.

Sind solche Korrekturen durchgeführt worden, so sollte man mit FLUSH den Inhalt des geänderten Textfeldes auf Diskette schreiben.

Neben diesen Befehlen für die Korrektur von Zeichen, gibt es noch Befehle zur Korrektur von Zeilen. Ist mit P ein Text in den Eingabespeicher übernommen worden, so ist er auch nach Aufruf einer anderen Zeile z. B. mit T 10 noch enthalten und kann dort mit

P⟨CR⟩

eingesetzt werden.

Beispiel:

```
T 3⟨CR⟩
P HALLO⟨CR⟩
T 10⟨CR⟩
P⟨CR⟩
```

Durch diese Befehlsfolge wird die Zeichenkette HALLO sowohl nach Zeile 3 und nach Zeile 10 geschrieben.

Durch das Wort U kann unterhalb der augenblicklichen Zeile eine neue Zeile eingefügt werden. Dabei werden alle bisherigen Zeilen eine Stelle tiefer geschoben. Der Inhalt der Zeile 15 geht somit verloren.

Durch

```
T 5⟨CR⟩
U GUTEN TAG⟨CR⟩
```

wird eine neue Zeile 6 mit dem Text GUTEN TAG geschaffen.
Diese Zeile kann durch X wieder gelöscht werden. Alle darunterliegenden Zeilen werden eine Zeile höher geschoben. Die Zeile 15 enthält Leerzeichen.

Damit soll die Beschreibung des Editors abgeschlossen werden, da durch die Wandlungsfähigkeit von FORTH die meisten Editoren für die verschiedenen FORTH-Versionen einen erweiterten oder geänderten Befehlssatz haben.

5.3. Zusammenstellung der Definitionen (nur für Textfelder, nicht für Editor)

((n1 n2 n3 -- ⟩ n1 n2 n3)

Kommentarbeginn. Text wird bis)übergangen.

LIST (n1 n2 n3 -- ⟩ n1 n2)

Ausgabe von Textfeld n3 auf Bildschirm.

FLUSH (n1 n2 n3 -- ⟩ n1 n2 n3)

Speichern des augenblicklichen Textfeldes.

LOAD (n1 n2 n3 --> n1 n2)

Compiliere Textfeld n3

CLEAR (n1 n2 n3 --> n1 n2)

Lösche Textfeld n3

COPY (n1 n2 n3 --> n1)

Kopiere Textfeld n2 nach n3

> (n1 n2 n3 --> n1 n2 n3)

Text geht im folgenden Textfeld weiter.

;S (n1 n2 n3 --> n1 n2 n3)

Ende für Compilierung

UPDATE (n1 n2 n3 --> n1 n2 n3)

Kennzeichnung des augenblicklichen Textfeldes als geändert.

NOTIZEN

6 Das Wörterbuch

6 Das Wörterbuch

In diesem Kapitel soll der Aufbau des Wörterbuches etwas näher untersucht werden. Gleichzeitig soll gezeigt werden, wie FORTH eine Definition aufbaut (COMPILE TIME Verhalten) und wie dies dann als Programm abläuft (RUN TIME Verhalten).

Zuerst soll aber die Speicherplatzverteilung eines FORTH Systems am Beispiel des poly-FORTH gezeigt werden.

6.1. Speicherbelegung

Die Speicherbelegung zeigt Abbildung 6.1.

Im unteren Speicherbereich ist der vorübersetzte Teil von FORTH. Dieser enthält alle, zum Programmablauf nötigen Definitionen entweder als Maschinencode oder als Wort, unter Benutzung der bisherigen Definitionen. Sind diese Abläufe zeitkritisch, wie z. B. das Durchsuchen des Wörterbuches, so sind diese Teile in Maschinencode programmiert.

Danach folgen die vom System verwendeten Variablen. Einige dieser Variablen werden wir in diesem Kapitel kennenlernen.

Danach folgen weitere Definitionen, die im Augenblick gerade benötigt werden. Dies kann zum Beispiel das Wörterbuch des Editors oder des Assemblers sein. Auf diesen Stapel wird dann das Wörterbuch des Benutzers aufgebaut. Eine Systemvariable HERE zeigt immer auf den nächsten freien Platz im Wörterbuch. Dises obere Ende des Wörterbuch-Stapel soll mit TOD (Top of Dictionary) bezeichnet werden.

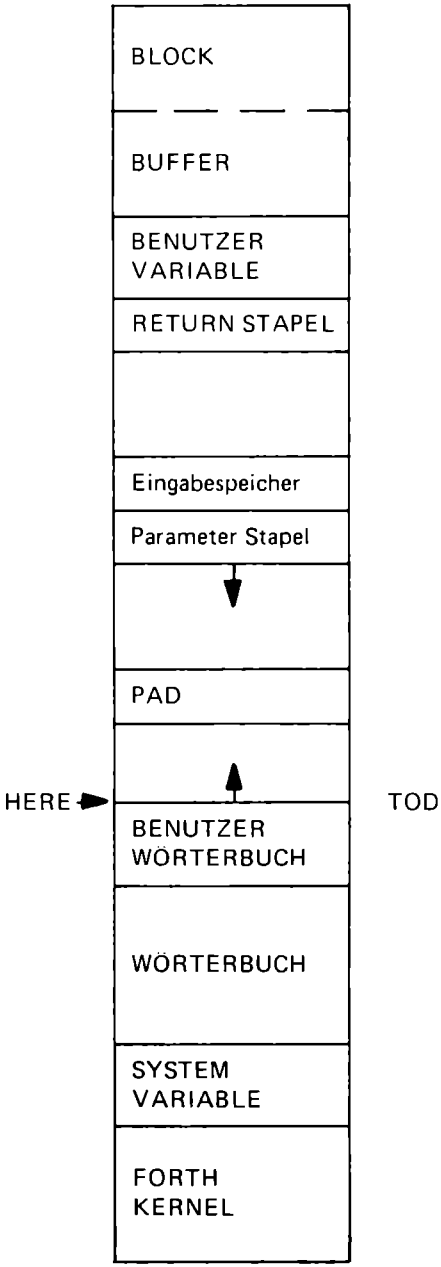


Abbildung 6.1
Speicherbelegung von FORTH

PAD ist ein Speicherbereich zu Aufnahme von Text mit maximal 64 Zeichen. Der Abstand zwischen TOD und PAD ist immer gleich.

Oberhalb PAD ist der Daten- oder Rechenstapel. Dieser wächst in Richtung kleiner werdenden Adressen. Die Systemvariable S0 enthält den Stapelanfang, während 'S (Tick S) die Adresse angibt, bis zu welcher sich der Stapel aufgebaut hat.

Direkt über den TOS beginnt der Eingabe-Speicher. In diesen werden die vom Tastenfeld eingegebenen Zeichen eingelesen. Der Eingabespeicher wächst in Richtung größer werdender Adressen. Gegen diesen Speicherbereich wird der RETURN-Stapel in Richtung abnehmender Adressen aufgebaut.

Über dem RETURN-Stapel werden die vom Benutzer definierten Variablen gespeichert und dann folgen Datenspeicher und Blöcke.

Dies ist die Verteilung des Speichers, wie in poly-FORTH in einer LSI-11 verwendet. Andere Beispiele für Speicherbelegung werden später noch angegeben werden. Die Namen der Systemvariablen sind aber gleich.

6.2. Ändern des TOD

Im folgenden sollen nun Änderungen im TOD vorgenommen werden. Um Herauszufinden welche Adresse die erste freie Speicherzelle im Wörterbuch hat, geben wir

```
HERE .
```

ein, und erhalten die Zahl 23964. Diese Zahl wird je nach Rechner und bisher aufgebauten Wörterbuch immer verschieden sein.

Deshalb sind die Zahlenangaben in den folgenden Beispielen nur als symbolische Adressen anzusehen. (Verwendeter Rechner APPLE II mit FORTH-79.)

```
HERE . 23964  OK
```

Nun geben wir eine Definition

```
: ADD + ;   OK
```

ein, und erhalten dann mit

```
HERE . 23976   OK
```

Die Definition ADD belegt 14 Byte in unseren Wörterbuch. Den Auf bau des Wortes wollen wir uns mit einem DUMP ansehen.

```
23964 15 DUMP

ADDR: 5D9C
5D98  43  E A3  D 83 41 44 C4  C.#..ADD
5DA0  3E 5D B6  D 9C  C A3  D  >]6...#.
5DA8   4 44 55 4D 50  0 20  0  .DUMP. .
 OK
```

Diese Zahlen ordnen wir wie in Abbildung 6.2 gezeigt an.

HERE nach der Definition →	5DA8			23976
Parameter Feld	5DA6	A3	0D	23974
	5DA4	9C	0C	23972
Code Feld	5DA2	B6	0D	23970
Link Feld	5DA0	3E	5D	23968
Namen Feld	5D9E	44	C4	23966
HERE vor der Definition →	5D9C	83	41	23964

Abbildung 6.2
Aufbau eines Wortes : ADD + ;

Zuerst kommt das Namenfeld. Das erste Byte gibt die Länge des Namens an. Dann folgen die ASCII-Zeichen des Namens.
In unserem Beispiel haben wir ein Wort, ADD, mit 3 Buchstaben verwendet. Damit erhalten wir in FORTH-79 oder poly-FORTH die gleiche Länge für den Wortaufbau.

In beiden FORTH-Versionen können Namen bis zu 31 Zeichen lang sein. poly-FORTH speichert aber nur die Länge und die ersten 3 Buchstaben ab.
Nach dem Namenfeld folgt das Link-Feld. Dort ist die Adresse der letzten Definition gespeichert, die vor dem ADD gemacht wurde. Das Link Feld wird auch als Verbindungsfeld bezeichent.
Im Code-Feld ist die Adresse gespeichert, bei der die Ausführung des Wortes beginnt. In diesem Falle ist es die Adresse der : Definition.

Als nächstes folgt das Parameter-Feld. Dort sind die Codefeldadressen der Worte angegeben, die in der Definition nach dem Namen angeführt sind. In unserem Fall ist dies die Adresse für die Definition von +, gefolgt von der Adresse für ; . Bei der Ausführung von ADD erfolgt mit dieser Adresse der Aufruf der nächsten Definition.

Um dies noch etwas zu verdeutlichen, wollen wir das Wort ADD bei der Definition eines neuen Wortes #AD verwenden.

```
: #AD DUP ROT ADD ;
```

Der Inhalt der Zellen 23976 bis 23992 ist in Abbildung 6.3 angegeben.

Im Namenfeld ist 83 # AD gespeichert. Das Link-Feld enthält die Adresse 5D9C. Dies ist die Adresse des Namenfeldes ADD. Das Code Feld enthält wiederum die Adresse der : Definition. Dann folgen die Adressen für DUP und ROT. Für das Wort ADD wird im Parameterfeld von #AD die Code-Feld-Adresse 5DA2 gespeichert.

Wird nun das Wort #AD aufgerufen, so wird zuerst die : Definition, dann DUP und ROT aufgerufen. ADD führt dann zur nochmaligen Ausführung von : und damit zum Aufruf von +. Der ; in ADD führt zurück zu ; in #AD. Damit ist der Aufruf von #AD beendet.

```
23964 40 DUMP

ADDR: 5D9C
5D98  43  E A3  D 83 41 44 C4  C.#..ADD
5DA0  3E 5D B6  D 9C  C A3  D  >]6...#.
5DA8  83 23 41 C4 9C 5D B6  D  .#AD.]6.
5DB0  A4  A 23 11 A2 5D A3  D  $.#."]#.
5DB8   4 44 55 4D 50  0 20 20  .DUMP.
5DC0   0  0  0  0  0  0  0  0  ........
 OK
```

HERE nach der Definition →	5DB8			23992
;	5DB6	A3	0D	23990
ADD	5DB4	A2	5D	23988
ROT	5DB2	23	11	23986
DUP	5DB0	A4	0A	23984
: Code Feld	5DAE	B6	0D	23982
Link Feld	5DAC	9C	5D	23980
Namen Feld	5DAA	41	C4	23978
HERE vor der Definition →	5DA8	83	23	23976

Abbildung 6.3
Aufbau des Wortes: #AD DUP ROT ADD ;

Ist die Codefeld-Adresse eines Wortes bekannt, so kann dieses Wort durch EXECUTE zur Ausführung gebracht werden. Die Codefeld-Adresse des + Zeichens ist in unserem Beispiel 0C9C Hex gleich 3228 Dezimal. Mit.

```
1 2 3228 EXECUTE . 3  OK
```

wird die Addition zweier Zahlen ausgeführt.

6.3. Speicherplatzreservierung

Bei den bisherigen Definitionen wurde kein zusätzlicher Speicherplatz benötigt. Will man aber Matrizen verarbeiten oder Messdaten aufzeichnen, so muß man sich Platz für diese Daten reservieren. Die Definition für diese Reservierung soll nicht sofort gemacht werden. Vorher sollen einige Beispiele gebracht werden, die zeigen wie man diese Reservierung definiert.

Wir benutzen ein neues Wort <BUILDS , das folgendes bewirkt:

```
<BUILDS VECTOR
```

legt im Wörterbuch ein neues Wort VECTOR an, das aus dem Namensfeld, dem Verbindungsfeld und dem Codefeld besteht. Das Parameterfeld ist leer und mit HERE wird diese Adresse angegeben. Damit die nächste Definition nicht mit dieser Adresse beginnt, verschieben wir den HERE Zeiger zu einer höheren Adresse. Die Definition von HERE ist

```
: HERE DP @ ;
```

Es wird also eine Systemvariable DP (der Name ist bei den einzelnen FORTH-Versionen verschieden) verwendet.
Mit

```
DP @ 10 + DP !   OK
```

wird der Zeiger HERE um 10 Byte verschoben. In DP ist also die Adresse der nächsten freien Stelle im Wörterbuch gespeichert. Diese Adresse wird um 10 erhöht.

Wird durch eine ! Operation eine Zahl in das Wörterbuch eingetragen, so muß der Zeiger HERE um 2 erhöht werden, da dies dann die nächste freie Stelle im Wörterbuch ist.

Die Wortfolge:

```
HERE ! DP @ 2+ DP !
```

wird durch das Wort , abgekürzt.

Die Definition von , ist

```
: , HERE ! DP @ 2+ DP ! ;
```

oder unter Verwendung des gleich beschriebenen Operators +!

```
L: , HERE ! 2 DP +! ;
```

Mit 1000 , 2000 , 3000 ,

werden die Zahlen 1000, 2000 und 3000 ins Wörterbuch übertragen. Der Zeiger HERE verweist dann auf die nächste freie Stelle hinter der Zahl 3000.

Für die Befehlsfolge

```
<NAME> @ N + <NAME> !
```

gibt es eine abgekürzte Schreibweise +! , mit der Definition:

+! (n1 n2 adr -- n1)

Addiert n2 zum Inhalt einer Variablen, deren Adresse adr auf dem Stapel ist.

Mit

```
10 DP +!
```

verschieben wir den Zeiger HERE um 10 Byte nach oben.

Um nun Zahlen in diesen freien Platz speichern zu können benötigen

wir die Adresse des Parameterfeldes. Diese wird uns durch den ' (Tick) Operator angegeben.

' ⟨NAME⟩ (n1 n2 -- ⟩ n1 n2 pfa)

Holt Parameterfeldadresse von ⟨NAME⟩ auf Stapel.

Damit können wir in diese Adresse eine Zahl speichern:

```
1024 ' VECTOR !
```

und mit

```
' VECTOR @
```

wieder zurückholen.

```
SCR # 81
  0  ( SPEICHERPLATZ RESEVIERUNG )
  1
  2  <BUILDS VECTOR DP @ 10 + DP !
  3  : @VECTOR ( NR -> N )
  4    ' VECTOR SWAP 2* + @ ;
  5  : !VECTOR ( N NR -> - )
  6    ' VECTOR SWAP 2* + ! ;
  7
  8
  9
 10
 11
 12
 13
 14    ;S
 15
```

Abbildung 6.4
Speicherplatzreservierung

In Abbildung 6.4 sind zwei neue Worte @VECTOR und !VECTOR definiert worden. Mit @VECTOR wird der Inhalt eines Speicherplatzes im reservierten Speicherbereich geholt. Die Nummer des Speicherplatzes NR ist auf dem Stapel. Da die Operation ! immer 2 Byte abspeichert, wird NR mit 2 multipliziert bevor es zur Adresse PFA addiert wird.

Das Speichern einer Zahl in diesen Bereich geschieht durch !VECTOR Hier wird auf dem Stapel die Zahl n und die Nummer NR des Speicherplatzes übergeben. Eine kurze Zwischenbemerkung:
Für die Namensgebung wurde hier eine in FORTH übliche Form verwendet. Viele einstelligen Worte bezeichnen einen bestimmten Vorgang, wie zum Beispiel

. Drucke, ! Speichere oder @ Hole.

Diese Zeichen kann man als ersten Buchstaben in einem Namen verwenden, um damit zu kennzeichnen was dieses bewirken soll. Wenn wir den Inhalt eines Speicherplatzes auf den Bildschirm ausgeben wollen, definierten wir ein Wort .VECTOR als

```
.VECTOR @ VECTOR . ;
```

Das Zeichen ' wird benutzt, um die Adresse einer Konstanten oder die Codefeldadresse eines Wortes zu finden. Das noch nicht besprochene Wort 'S wird sicher die Bedeutung haben, eine Adresse des Datenstapels zu finden.

Durch die Einführung solcher Bezeichnungen und Namen wird die Lesbarkeit von FORTH Programmen erheblich verbessert (siehe Anhang B).
Nun wieder zum Speichern von Daten in dem reservierten Speicherbereich.

In dem reservierten Feld soll in einem Speicherplatz vermerkt werden an welche Stelle die nächste Zahl geschrieben wird. Dazu verwenden wir die Definitionen in Abbildung 6.5.

```
SCR # 82
  0  ( INDIZIERTE SPEICHERUNG )
  1
  2  <BUILDS VECTOR  12 ALLOT
  3  : INIT ( * 1->PFA *) 1 ' VECTOR
  4  ! ;
  5  : +INDEX 1   VECTOR @ + ' VECTO
  6 R ! ;
  7  : @VECTOR 2*   VECTOR + @ ;
  8  : !VECTOR ' VECTOR DUP @
  9  2* + ! +INDEX ;
 10
 11
 12
 13
 14   ;S
 15
```

Abbildung 6.5
Indizierte Adressierung

In Zeile 2 wird ein neues Wort ALLOT eingeführt. Die Definition von ALLOT ist:

```
: ALLOT (  N --> ) DP @ + DP ! ;
```

Damit wird für n Byte Speicherplatz reserviert und somit können hier n / 2 Zahlen gespeichert werden.

Durch das Wort INIT wird in den untersten Speicherplatz als Anfangswert eine Eins gespeichert. Durch +INDEX wird diese Zahl nach jedem Abspeichern einer Zahl um Eins erhöht.

@VECTOR und !VECTOR sind die Worte zum Holen oder zum Speichern einer Zahl.

Programmablauf:

```
INIT OK
1000 !VECTOR OK
2000 !VECTOR OK
3000 !VECTOR OK
1 @VECTOR . 1000 OK
```

Dieses Beispiel wurde deshalb so ausführlich gebracht, um zu zeigen, wie in FORTH mit virtuellem Speicher gearbeitet wird.

Zuvor werden aber noch vier Worte eingeführt, welche zur Umrechnung von Parameterfeld, Codefeld, Verbindungsfeld und Namenfeld verwendet werden können.

Wird mit dem ' (Tick) Operator die Parameterfeldadresse auf den Stapel geholt, so kann sie mit CFA in die Codefeldadresse umgerechnet werden.

CFA (pfa --> cfa)

Soll aus der Parameterfeldadresse die Namenfeldadresse bestimmt werden, so wird dazu das Wort NFA verwendet.

NFA (pfa --> nfa)

Die Umrechnung der Namensfeldadresse in die Parameterfeldadresse erfolgt durch das Wort PFA.

PFA (nfa --> pfa)

Schließlich kann aus der Parameterfeldadresse durch LFA die Verbindungsfeldadresse bestimmt werden.

LFA (pfa --> lfa)

6.4. Virtueller Speicher

Im letzten Beispiel wurde durch das Wort 12 ALLOT Speicherplatz für die Anzahl der gespeicherten Zahlen und Platz für 5 Zahlen geschaffen. Was ist zu tun, wenn man große Zahlenmengen z. B. 10000 16 bit Zahlen speichern möchte. Wie wir im 5. Kapitel schon gesehen haben, sind die Blöcke im Speicher und auf der Diskette völlig gleichberechtigt. Wir werden also unsere Zahlen in diese Blöcke speichern. Dabei werden die Zahlen in den Blockpuffer übernommen, und dieser wird, wenn er voll ist, auf Diskette gespeichert.

Das Lesen einer Zahl geschieht auf folgende Weise:
Aus dem Index wird die Nummer des Blocks und die Adresse innerhalb des Blockes berechnet. Dieser Block wird in den RAM-Speicher gelesen und aus diesem Block die entsprechende Zahl ausgelesen. Der Benutzer selbst merkt kaum einen Unterschied, ob diese Zahl aus dem RAM oder von der Diskette geholt wird.
Die Diskette ist die gedachte (virtuelle) Erweiterung des RAM-Speichers.

Das Programm dazu zeigt Abbildung 6.6.

```
SCR # 85
  0  ( VIRTUELLER SPEICHER )
  1  400 CONSTANT START
  2  : INDEX ( N -> ADR ) 2* 128
  3  /MOD START + BLOCK + ;
  4
  5  : ANZAHL 0 INDEX ;
  6  : +NR ( NR --> NR+1 ) 1 ANZAHL
  7    +!  UPDATE ;
  8  : !SPEICHER ANZAHL @ INDEX !
  9  UPDATE +NR ;
 10  : @SPEICHER INDEX @ ;
 11  : INIT 1 ANZAHL ! UPDATE ;
 12  : .INHALT CR ANZAHL @ DUP 1 =
 13  IF ELSE 1 DO I @SPEICHER
 14  . CR LOOP THEN ;
 15
```

Abbildung 6.6
Vitueller Speicher

Für die Programmierung eines virtuellen Speichers ist die Kenntnis des Blockaufbaus der benutzten FORTH-Version notwendig. Das Programm in Abbildung 6.6. benutzt eine FORTH-Version mit einer Blocklänge von 128 Byte. Die Berechnung der Adresse für einen Eintrag mit gegebenen Index erfolgt mit dem Wort INDEX.

Der Index n wird mit zwei multipliziert und durch die Blocklänge /MOD geteilt. Damit ist nun auf dem Stapel der Quotient der Division und der verbleibende Rest.

Der Quotient gibt die Nummer des Blockes und der Rest die Lage der Zahl innerhalb des Blockes an. Auf einer Diskette sind sehr viele Blöcke mit FORTH Programm und FORTH Definitionen belegt. Für den Beginn der Speicherung von Daten wird man also einen der ersten freien Blöcke auf der Diskette verwenden. Deshalb wird zu der berechneten Blocknummer ein fester Betrag, START, hinzuaddiert.

Nach dieser Addition ist die Blocknummer auf dem Stapel. Das Wort BLOCK liest diesen Block von der Diskette und legt die Anfangsadresse auf dem Stapel ab. Zu dieser Adresse wird der Rest der /MOD Division addiert. Damit ist die Adressberechnung für die gesuchte Zahl mit gegebenen Index beendet.

Das Wort ANZAHL liest den Inhalt des Speicherplatzes mit dem Index 0. Dort ist die Anzahl der bereits in die Blöcke geschriebenen Daten gespeichert.

Mit +NR wird diese Zahl nach jedem Abspeichern um Eins erhöht.
Das Wort UPDATE bewirkt, daß der bearbeitete Block als verändert markiert wird und damit auf die Diskette zurückgeschrieben wird.

Es folgen die Worte !SPEICHER und @SPEICHER. Diese haben die gleiche Bedeutung wie die Worte !VECTOR und @VECTOR im vorangegangenem Beispiel.

Das Wort INIT setzt den Anfangswert von ANZAHL auf Eins.

Das Wort .INHALT gibt den gesamten gespeicherten Inhalt auf den Bildschirm aus.

Um nun nicht viele Zahlenwerte eintippen zu müssen, verwenden wir das ! SPEICHER Wort in einer Schleife.

```
: TEST 1 DO 6566 !SPEICHER LOOP ;
```

Mit

```
INIT 100 TEST
```

schreiben wir 99 mal die Zahl 6665 in den angewählten Block.

Beim Arbeiten mit diesem Programm ist Vorsicht geboten. FORTH legt die gesamte Verantwortung für die Speicherung von Daten in die Hand des Programmierers. Es erfolgt keine Fehlermeldung bei Überschreiten von Indexgrenzen, denn es gibt keine. Ebensowenig wird geprüft, ob diese Daten in einen bereits beschriebenen Block gespeichert wurden. Bei einer falschen Wahl der Blocklänge ist es durchaus möglich, das Blöcke, die FORTH-Programm enthalten, überschrieben werden, was zu einem Systemzusammenbruch führen kann. Deshalb sollte dieses Programm beim Testen mit einer Kopie des FORTH-Programm vorgenommen werden.

Diese Technik des virtuellen Speichers ist bestens geeignet, große Datenmengen zu speichern und zu verarbeiten. Darüber hinaus dürfte aufgefallen sein, daß keine der in anderen Sprachen üblichen Definitionen für das Eröffnen und Schließen von Datenfiles notwendig sind. Auch die Angabe von Recordlänge, speziellen Parametern und ähnlichen Befehlen ist nicht notwendig. Diese Blöcke sind einfach vorhanden und verhalten sich wie RAM. Daten können eingeschrieben und wieder gelesen werden.

Als Programmierübung soll ein Wort definiert werden, das bei gegebenen Index die entsprechende Zahl liest, vorher aber prüft, ob an diese Stelle schon etwas gespeichert wurde.

6.5. Aufstellen eines eigenen Wörterbuchs

Im Normalfall werden alle neu definierten Worte in das FORTH-Wörterbuch eingetragen. Zwei Systemvariable CURRENT und CONTEXT enthalten eine Adresse die auf das FORTH Wörterbuch verweist. Dabei enthält CONTEXT die Adresse des Wörterbuchs, in welchem ein Wort zuerst gesucht wird, und CURRENT die Adresse des Wörterbuchs, in welches ein Wort eingetragen wird. In den meisten Fällen wird CURRENT gleich CONTEXT sein.

Durch das Wort VOCABULARY ⟨NAME⟩ wird nun auf den Wörterbuchstapel der Beginn eines neuen Wörterbuch eingetragen. Durch ⟨NAME⟩ DEFINITIONS wird CURRENT auf CONTEXT gesetzt und danach werden alle weiteren Definitionen in das neue Wörterbuch eingetragen.

Beispiel:
Wir definieren in FORTH ein Wort WO, das uns den Text BIN IN FORTH ausgibt:

```
: WO ." BIN IN FORTH " ;  OK
```

Nun eröffnen wir ein neues Wörterbuch TEST und tragen neue Definitionen dort ein.

```
VOCABULARY TEST  OK
TEST DEFINITIONS  OK
```

Hier definieren wir nochmals das Wort WO, mit dem Text: BIN IN TEST .

```
: WO ." BIN IN TEST " ;
WO NOT UNIQUE   OK
```

Wir erhalten zwar die Warnung WO NOT UNIQUE, da bei der Eintragung auch im FORTH-Wörterbuch gesucht wird, aber wenn wir jetzt WO aufrufen, erhalten wir die Meldung

```
WO  BIN IN TEST  OK
```

Mit dem Wort FORTH setzen wir CONTEXT auf das FORTH-Wörterbuch. Wenn wir jetzt WO aufrufen, erhalten wir die Meldung:

```
WO  BIN IN FORTH  OK
```

Die Variablen CONTEXT und CURRENT enthalten jetzt verschiedene Adressen. CONTEXT enthält die Adresse des FORTH-Wörterbuchs, und somit wird dort zuerst nach einem Wort gesucht. CURRENT enthält die Adresse von TEST, da dort die Definitionen abgelegt werden.

Die Definition

```
: WOHIN ." NACH TEST " ;    OK
```

Wird im Wörterbuch TEST abgelegt. Allerdings kann man, wenn man ein eigenes Wörterbuch angelegt hat, das FORTH-Wörterbuch nicht mehr erweitern.

Nach Vereinbarung sollten eigene Wörterbücher als IMMEDIATE gekennzeichnet werden. Dies bedeutet, daß das Wort immer sofort, auch während des Compilierens sofort ausgeführt wird.
Auf die Bedeutung von IMMEDIATE wird im übernächsten Abschnitt eingegangen werden.

In den meisten FORTH-Versionen ist das FORTH-Wörterbuch gegen das Löschen von Einträgen geschützt. Die Eingabe von FORGET VLIST gibt die Warnung VLIST ? PROTECTED aus. Das bedeutet, daß VLIST im geschützten Teil des Wörterbuches liegt.
Eine Systemvariable FENCE enthält die Namensfeldadresse des ersten Eintrags im ungeschützten Bereich. Dies ist vor diesem Eintrag der Zeiger HERE. Damit kann man durch

```
HERE FENCE !
```

eigene Einträge vor dem Vergessen schützen. Diesen Schutz kann man mit

```
' <NAME> NFA FENCE !
```

aufheben.

Mit ' ⟨NAME⟩ wird die Parameterfeldadresse des Eintrags ⟨NAME⟩ geholt und durch NFA die zugehörige Namensfeldadresse berechnet und diese in FENCE gespeichert. Damit ist dieser Eintrag wieder ungeschützt und kann somit gelöscht werden.

6.6. Definitionsworte

Mit dem Begriff Definitionsworte sollen solche Worte in FORTH bezeichnet werden, die ein neues Wort definieren.

Dies sind : ⟨BUILDS (CREATE in poly-FORTH) CONSTANT und VARIABLE.

Auf die Definition eines Definitionswortes mit ⟨BUILDS soll jetzt näher eingegangen werden. In Kapitel 6.2 haben wir dieses Wort benutzt um bei der Übersetzung oder Compilierung den Kopf eines Wortes ohne Parameterfeld in das Wörterbuch einzutragen. Wir haben dann andere Worte benutzt um Daten im Parameterfeld abzulegen. Diese beiden getrennten Definitionen können zu einer einzigen Definition zusammengezogen werden. Den Aufbau einer solchen Definition für ein Definitionswort zeigt Abbildung 6.7. Zuerst wird der Name des neuen Definitionswortes festgelegt und nach dem Wort ⟨BUILDS wird die Folge von Definitionen angegeben, die das Verhalten während der Compilierung festlegen. Nach dem Wort DOES⟩ wird dann die Folge von Definitionen angegeben, die das Verhalten während der Laufzeit angeben.

Als Beispiel dazu die Definition des Definitionswortes CONSTANT

```
: CONSTANT <BUILDS , DOES> @ ;
```

Abbildung 6.7
Definition eines Definitionswortes

Das Wort mit dem Namen CONSTANT definiert eine Klasse von Worten, die bei der Compilierung eine Zahl im Wörterbuch speichern ( ) und bei der Ausführung eines Programms diese aus dem Wörterbuch auslesen und auf dem Stapel ablegen.

Beim Aufruf von 3 CONSTANT DREI wird bei der Compilierung unter dem Namen DREI die Zahl 3 im Wörterbuch eingetragen. Wird bei der Ausführung eines Programms, also während der Laufzeit, das Wort DREI aufgerufen, dann wird die Zahl 3 aus dem Wörterbuch geholt und auf dem Stapel ausgelegt.

Auf die gleiche Weise können wir uns jetzt ein Definitionswort VECTOR definieren, das uns bei der Compilierung Speicherplatz für z. B. 5 Zahlen im Wörterbuch freihält. Bei der Ausführung von Programmen kann man dann unter Angabe eines Index dort Zahlen speichern oder auslesen. Die Definition lautet:

```
: VECTOR <BUILDS 20 ALLOT DOES> SWAP 2* + ;
```

Mit dem Definitionswort VECTOR definieren wir ein neues Wort FELD, das ein Zahlenfled von 10 Zahlen darstellt:

```
VECTOR FELD
```

Durch 1000 0 FELD wird in den untersten Speicherplatz die Zahl 1000 gespeichert. Mit

```
0 FELD ?
```

wird sie wieder auf den Bildschirm ausgegeben. Alle Namen die mit dem Definitionswort VECTOR definiert worden sind haben alle die gleiche Eigenschaft. Sie gehören alle zur gleichen Klasse. In unserem Fall wird bei der Compilierung Platz für 10 Zahlen freigehalten, in der während der Laufzeit Zahlen eingeschrieben oder ausgelesen werden können. Aber Vorsicht ! Es findet keine Überprüfung der Grenzen des Feldes statt. Der Indexbereich reicht von 0 bis 9. Wird ein

```
9999 10 FELD !
```

ausgeführt, so wird die Zahl 9999 in den Kopf der folgenden Definition geschrieben und somit das Wörterbuch zuerstört.

Falls man lieber den Indexbereich von 1 bis 10 verwenden will, so ändert man die Definiton in

```
: VECTOR <BUILDS 20 ALLOT DOES> 1- 2* + ;
```

ab.

Programmierübung:
Es soll ein Definitionswort ARRAY definiert werden, das für 10 Zahlen Speicherplatz freilässt. Beim Laufzeitverhalten werden die Grenzen überprüft und bei Ausgabe eines falschen Index die Fehlermeldung "FALSCHER INDEX" ausgegeben.

Das gleiche für ein Definitionswort VARRAY, bei dem die Grenzen während der Definiton des Wortes vorgegeben und bei der Laufzeit überprüft werden.

Die Schaffung von Definitionsworten stellt einen Eingriff in den Compiler dar. Durch die ⟨BUILDS . . . DOES⟩ Anweisung werden neue Datenstrukturen geschaffen und in den Compiler übertragen. Diese Änderung und Erweiterung des Compilers ist in FORTH durchaus erwünscht und erlaubt. Andere Sprachen versiegeln dagegen den Compiler gegen gewünschte Änderungen eines Programmierers. Diese Freizügigkeit setzt aber voraus, daß der Programmierer sich über die gemachten Änderungen im klaren ist und sie selbst auf Fehler prüft.

Das folgende Programm eines Sprachübersetzers in Abbildung 6.8 soll das Verhalten eines Wörterbuchs in FORTH nochmals aufzeigen.

In Zeile 1 wird das Definitionswort LISTE eingeführt, das Speicherplatz für 4 Zahlen reserviert. Diese Zahlen können durch Angabe eines Index gelesen oder eingeschrieben werden. Mit dem Definitionswort LISTE definieren wir das Wort 'LISTE (Adressenliste). Wir bezeichnen mit EN HELLO DE GUTEN TAG usw. und speichern die entsprechende Adressen ('EN 'DE . .) in unserer Liste.

```
SCR # 88
  0   ( SPRACHUEBERSETZER )
  1   : 'LISTE <BUILDS 8 ALLOT DOES>
  2   SWAP 2* + ; 'LISTE LISTE
  3
  4   : EN ." HELLO " ;
  5   ' EN 0 LISTE !
  6   : DE ." GUTEN TAG " ;
  7   ' DE 1 LISTE !
  8   : IT ." BON GIORNO " ;
  9     IT 2 LISTE !
 10   : FR ." BON JOUR " ;
 11   ' FR 3 LISTE !
 12
 13
 14     -->
 15

SCR # 89
  0   ( SPRACHUEBERSETZER )
  1
  2   0 CONSTANT ENGLISCHE
  3   1 CONSTANT DEUTSCHE
  4   2 CONSTANT ITALIENISCHE
  5   3 CONSTANT FRANZOESISCHE
  6
  7   : GRUESSE LISTE @ 2- EXECUTE ;
  8


 OK
 ENGLISCHE GRUESSE HELLO OK
 ITALIENISCHE GRUESSE BON GIORNO OK
 FRANZOESISCHE GRUESSE BON JOUR OK
```

Abbildung 6.8
Sprachübersetzer

Damit wird im FORTH Wörterbuch ein anderes Wörterbuch aufgebaut.

Mit 0 CONSTANT ENGLISCHE
1 CONSTANT DEUTSCHE usw.

weisen wir dem Wort ENGLISCHE die Zahl 0, dem Wort DEUTSCH die Zahl 1 usw. zu. Das Wort GRUESSE holt aus der LISTE die Adresse, die mit EXECUTE zur Ausführung gebracht wird.

Der Programmablauf in diesem Wörterbuch entspricht genau dem Programmablauf im FORTH Wörterbuch. In einer Tabelle (Parameterfeld) sind die Adressen von Worten (Programm) gespeichert. Diese Programmteile werden durch indirekte Sprünge zur Ausführung gebracht.

Ein weiteres Definitionswort ist CREATE.

CREATE ⟨NAME⟩

Erzeugt Eintrag in das Wörterbuch von ⟨NAME⟩ und setzt als Codefeldadresse die Parameterfeldadresse des Wortes.

Mit CREATE lassen sich kurze Maschinenprogramme eingeben.
Das folgende Beispiel ist die Initialisierung von Tor B des ATARI zur Vorbereitung auf eine Druckerausgabe.

```
SCR # 80
  0  ( SCREENPRINT )   HEX
  1  CREATE INIT   ( PORT B )
  2  A9 C, 30 C,   ( LDA #$30 )
  3  8D C, 03 C, D3 C, ( STA $D303 )
  4  A9 C, FF C,   ( LDA #$FF )
  5  8D C, 01 C, D3 C, ( STA $D301 )
  6  A9 C, 34 C,   ( LDA #$34 )
  7  8D C, 03 C, D3 C, ( STA $D303 )
  8  A9 C, 80 C,   ( LDA #$80 )
  9  8D C, 01 C, D3 C, ( STA $D301 )
 10  4C C, 47 C, 07 C, ( JMP NEXT   )
 11  SMUDGE
```

Abbildung 6.9
Wortdefinition mit CREATE

CREATE und ⟨BUILDS sind meistens gleichbedeutend. Durch CREATE wird Bit 6 im Zählbyte des Namens auf Eins gesetzt. Dadurch wird das Wort beim Suchen übergangen. Mit SMUDGE schaltet Bit 6 um.

4 / Ändern des Compile-Verhaltens

Bei einer Wortdefinition wie zum Beispiel:

```
: T 10 0 DO 65 EMIT LOOP ;
```

werden die Zahlen in der augenblicklich gültigen Zahlenbasis eingegeben. Ist Dezimalrechnung eingeschaltet, so sind die Zahlen Dezimalzahlen, ist Hexadezimalrechnung eingeschaltet, so sind es Hexadezimalzahlen. Ein nachträgliches Ändern der Zahlenbasis ändert nicht die Zahlen in der Definition. Dies kann umgangen werden, wenn wenn man den Compiler während der Eingabe aus- und einschaltet. Dies geschieht durch die Worte [und] .

[(-->)

Schaltet Compiler aus.

] (-->)

Schaltet Compiler ein.

Beispiel:

```
: T 10 0 DO [ HEX ] 2E EMIT [ DECIMAL ] LOOP ;
```

Durch diese Schleife sollen 10 Punkte ausgegeben werden. Vor Beginn der Definition ist auf Dezimalrechnung geschaltet worden. Die Zahlen 10 und 0 sind Dezimalzahlen. Der Punkt soll als ASCII-Zeichen 2E eingegeben werden. Dazu wird mit [der Compiler ausgeschaltet, das Wort HEX direkt ausgeführt und mit] der Compiler wieder eingeschaltet. FORTH befindet sich nun in Hexadezimalrechnung. Nach der Eingabe von 2E wird wieder mit [DECIMAL] zurückgeschaltet. Die zwischen den eckigen Klammern stehenden Worte werden während des Compilierens sofort ausgeführt.

Bei dem folgenden Beispiel werden die Worte VORHER und NACH-

HER während des Compilierens ausgegeben.

```
SCR# 102   : T [ ." VORHER " HEX ] 87 EMIT
               [ ." NACHHER" DECIMAL ]
               ; 102 LOAD VORHER NACHHER OK
```

Wird T ausgeführt, so wird nur das ASCII-Zeichen BELL ausgegeben.

Im allgemeinen wird eine Definition nur dann ausgeführt, wenn sie auf gerufen wird. Durch das Wort IMMEDIATE wird eine Definition sofort nach der Compilierung ausgeführt.

Beispiel:

```
: SOFORT ." SOFORT " ; IMMEDIATE  OK

: LOS SOFORT ." GEHTS LOS " ; SOFORT  OK
```

Das Wort SOFORT wurde als IMMEDIATE erklärt. Bei der Compilierung wird es demnach sofort ausgeführt. Wird das Wort LOS aufgerufen, erscheint nur die Meldung

```
LOS  GEHTS LOS  OK
```

Durch [COMPILE] ⟨NAME⟩ kann ein, als IMMEDIATE gekennzeichnetes Wort in die Definition compiliert werden.

```
: NLOS [COMPILE] SOFORT ." GEHTS LOS " ;

  NLOS  SOFORT GEHTS LOS  OK
```

Dazu noch ein Beispiel aus QS-FORTH

Das Wort LOCATE ⟨NAME⟩ sucht das Textfeld, in welchem das Wort ⟨NAME⟩ definiert wurde. Dazu trägt QS-FORTH bei von Disketten

compilierten Worten vor der Namenfeldaresse die Nummer des Textfeldes ein. Der ' Befehl ist IMMEDIATE, durch [COMPILE] wird er in die Definiton von LOCATE aufgenommen. Danach wird der Compiler abgeschaltet und der augenblickliche Wert von FENCE auf den Stapel geholt. LITERAL fügt nun diese Zahl in die Wortdefinition ein.

```
SCR # 1
  0     FORTH DEFINITIONS HEX
  1
  2   : L-ON  0780   CREATE ! ;
  3   : L-OFF ' FIRST CFA
  4             CREATE ! ; DECIMAL
  5   : LOCATE [COMPILE] ' DUP
  6          [ FENCE @ ] LITERAL >
  7     IF NFA 2 - @ DUP 0=
  8     IF     25 MESSAGE
  9     ELSE   DUP 0 <
 10     IF 5 MESSAGE QUIT ELSE LIST
 11     ENDIF ENDIF
```

Die Definition von LITERAL lautet:

LITERAL (n -->)

während des Compilierens wird eine Zahl vom Stapel in die Wortdefinition eingefügt.

(--> n)

während der Laufzeit wird diese Zahl auf dem Stapel abgelegt.

Wie schon vorher erwähnt, stellt in FORTH als Compiler kein isoliertes Programm dar. Es können Eingriffe in den Compiler gemacht und der Compiler selbst oder das Compiler-Verhalten verändert werden.

6.8. Zusammenstellung der Definitionen

HERE (n1 n2 --> n1 n2 adr)

Bringt die Adresse des nächsten freien Speicherplatzes auf den Stapel.

EXECUTE (n1 n2 adr --> n1 n2)

Bringt ein Programm, das bei adr anfängt zur Ausführung. adr ist die Codefeldadresse eines FORTH Wortes.

, (n1 n2 n3 --> n1 n2)

Speichert n3 in die Speicherzelle auf die HERE zeigt und erhöht den Zeiger um Zwei.

+! (n1 n2 adr --> n1)

Addiert n2 zum Inhalt einer Variablen, deren Adresse adr auf dem Stapel ist.

' ⟨NAME⟩ (n1 n2 --> n1 n2 pfa)

Bringt die Parameterfeldadresse von ⟨NAME⟩ in den TOS.

CFA (pfa --> cfa)

Rechnet die Parameterfeldadresse in die Codefeldadresse um.

NFA (pfa --> nfa)

Rechnet die Parameterfeldadresse in die Namenfeldadresse um.

PFA (nfa --> pfa)

Rechnet die Namenfeldadresse in die Parameterfeldadresse um.

LFA (pfa --> lfa)

Rechnet die Parameterfeldadresse in die Verbindungsfeldadresse um.

ALLOT (n1 n2 n3 --> n1 n2)

Reserviert n3 Byte Speicherplatz im Wörterbuch.

BUILDS ‹NAME› (n --> n)

Trägt ‹NAME› ins Wörterbuch ein. Belegt Namenfeld, Indexfeld und Codefeld, kein Parameterfeld.

DOES› (n --> n)

Wird in Definitionsworten benutzt. Beendet den Teil für die COMPILE-Anweisungen und zeigt den Beginn der Laufzeitanweisungen an.

Definitionswort:

: ‹NAME› ‹BUILDS ‹WORTE1› DOES› ‹WORTE2› ;

Bei der Compilierung eines Wortes ‹NAME› werden die ‹WORTE1› ausgeführt. Bei der Laufzeit dann ‹WORTE2›. Alle mit ‹NAME› definierten Worte bilden eine Klasse gleicher Eigenschaften.

CREATE ‹NAME›

Eintrag von ‹NAME› ins Wörterbuch. Codefeldadresse ist Parameterfeldadresse des Eintrags.

[(-->)

Schaltet Compiler aus.

] (-->)

Schaltet Compiler ein.

LITERAL (n -->)

Bei Compilierung n vom Stapel in Wortdefinition einfügen.

(--> n)

Bei Laufzeit n auf Stapel ablegen.

BLOCK (n --> adr)

Bringt die Adresse des Blockspeichers auf den Stapel, der den Block mit der Nummer n enthält. Ist dieser noch nicht im Speicher, wird er von der Diskette gelesen.

DP (--> adr)

Variable, enthält Adresse des ersten freien Platzes im Wörterbuch.

SMUDGE (-->)

Schaltet Bit 6 des Längenbytes im Namenseintrag um. Mit Bit 6 = 1 wird Name beim Suchen nicht gefunden.

7 Texteingabe

7 Texteingabe

In der Grundversion von FORTH sind keine Worte zur Verarbeitung von Zeichenketten vereinbart. Diese lassen sich aber leicht durch die für die Ein- und Ausgabe von Text vorhandenen Worte definieren.

Das Wort EXPECT erwartet die Eingabe von Text. Die Definition von EXPECT lautet

EXPECT (adr n -->)

Eingabe von n Zeichen. Text nach Adresse adr.

Dieser Text kann mit TYPE wieder ausgegeben werden (Definition Kapitel 4). Nach EXPECT muß, bei der direkten Eingabe ein CR eingegeben werden.

Beispiel:

```
PAD 10 EXPECT GUTEN TAG  OK

PAD 10 TYPE GUTEN TAG OK
```

Für die Eingabe von Text haben wir als Eingabespeicher PAD verwendet. PAD liegt normalerweise 68 Byte über HERE (siehe Abbildung 6.1).

Beim Aufruf von PAD wird die Anfangsadresse auf den Stapel abgelegt.

PAD (--> adr)

Speicherbereich 68 Byte über HERE.

Anstelle von PAD hätte man auch eine beliebige Adresse aus dem freien RAM-Bereich des Rechners nehmen können. Es ist allerdings sinnvoll diesen RAM-Speicher vorher mit Leerzeichen ($20) oder Nullen zu füllen. Dies kann mit FILL oder ERASE geschehen.

FILL (adr n c -->)

Füllt ab Adresse adr n Byte mit dem ASCII-Zeichen c.

ERASE (adr n -->)

Füllt ab Adresse adr n Byte mit 00.

Beispiel:

```
HEX 6000 40 20 FILL  OK

6000 20 EXPECT DIES IST TEXT  OK
6000 4 TYPE DIES OK
6000 2 + 5 TYPE ES IS OK
6005 5 TYPE IST T OK
```

Mit der folgenden Definition kann Text nach PAD eingegeben werden.

```
: TEXT ( C --> )
        PAD 72 32 FILL WORD HERE COUNT
        PAD SWAP CMOVE ;
```

Dabei werden folgende Worte verwendet:

WORD (c -->)

WORD übernimmt eine Zeichenfolge aus dem Eingabespeicher bis zum Begrenzerzeichen c.

dem Text wird durch WORD ein Zählbyte eingefügt, das die Länge des Textes bis zum Begrenzer angibt.

Einige FORTH Versionen haben eine andere Definition für WORD

WORD (c --> adr)

adr ist die Adresse des Zählbytes.

COUNT (adr --> adr+1 n)

COUNT legt auf dem Stapel die Länge und die Anfangsadresse des Textes auf dem Stapel ab. adr ist die Adresse des Zählbytes.

CMOVE (adr1 adr2 n -->)

CMOVE verscheibt n Byte von Adresse adr1 nach Adresse adr2.

Das bei adr1 gespeicherte Byte wird als erstes nach adr2 geschrieben. Danach werden die nächst höheren Byte umgesetzt. Wird durch CMOVE der Speicherinhalt zu höheren Adressen verschoben, so muß adr2 größer als adr1 + n sein. Ist dies nicht der Fall, so wird der Speicherinhalt verändert.

In der Definition TEXT werden zuerst 72 Speicherplätze von PAD mit Leerzeichen gefüllt. Danach wird durch WORD der Text auf das Begrenzerzeichen untersucht und ab der Adresse HERE+1 gespeichert.

Das Zählbyte selbst wird bei HERE abgelegt. Mit COUNT wird die Anfangsadresse des Textes und die Länge auf dem Stapel abgelegt. Nach PAD SWAP enthält der Stapel die richtige Reihenfolge für CMOVE. Vor dem Aufruf von TEXT muß das Begrenzerzeichen auf dem Stapel sein.

Beispiel:

```
13 TEXT DIES IST TEXT  OK

PAD 20 TYPE DIES IST TEXT        OK

32 TEXT DIES IST TEXT
IST ? UNKNOWN

PAD 20 TYPE DIES                 OK
```

Bei der ersten Eingabe wurde als Begrenzer das ASCII-zeichen für RETURN (13, HEX 0D) eingegeben. Dieser Text wurde auch vollständig übernommen. Bei der zweiten Eingabe wurde als Begrenzer das Leerzeichen (32, HEX 20) verwendet. Nach PAD wird deshalb nur das erste Wort DIES übernommen. Der Interpreter allerdings findet noch Text vor, und versucht diesen zu übersetzen. Er liest das nächste Wort bis zum Leerzeichen und sucht danach, ob er es im Wörterbuch finden kann. IST ist nicht definiert und somit erscheint die Fehlermeldung IST ?.

Das Programm in 7.1 zeigt die Eingabe einer Adresse nach PAD und die Ausgabe dieser Adresse auf den Bildschirm.

```
SCR # 78
  0  ( ADRESSEN EINGABE )
  1
  2  : NAME PAD 10 + 10 EXPECT ;
  3  : VORNAME PAD 10 EXPECT ;
  4  : STRASSE PAD 20 + 15 EXPECT ;
  5  : PLZ PAD 35 + 4 EXPECT ;
  6  : ORT PAD 39 + 15 EXPECT ;
  7  : EIN CR
  8  ." VORNAME " VORNAME
  9  ."     NAME " NAME     CR
 10  ." STRASSE " STRASSE CR
 11  ."      PLZ " PLZ
 12  ."      ORT " ORT      CR ;
 13
 14
 15  -->
```

```
SCR # 79
  0   ( ADRESSEN AUSGABE )
  1
  2   : 1L ."   " ;
  3   : 5L ."       " ;
  4   : PRINT ( ADR L -->  )
  5     PAD + SWAP -TRAILING TYPE ;
  6   : AUS CR 5L 10 0 PRINT 1L
  7     10 10 PRINT CR 5L 15 20 PRINT
  8     CR 4 35 PRINT 1L 15 39 PRINT
  9     CR ;
 10
```

Abbildung 7.1
Adressen Ein- und Ausgabe

7.1. Zusammenstellung der Definitionen

EXPECT (adr n -->)

Eingabe von n Zeichen nach Adresse adr.

PAD (--> adr)

Adresse des Speichers PAD auf Stapel.

FILL (adr n c -->)

Füllt ab Adresse adr n Byte mit ASCII-Zeichen c.

ERASE (adr n -->)

Füllt ab Adresse adr n Byte mit 00.

WORD (c -->)

übernimmt eine Zeichenfolge aus dem Eingabespeicher bis zum Begrenzer c.

COUNT (adr --> adr+1 n)

legt auf dem Stapel die Länge und die Anfangsadresse des Textes ab. adr ist die Adresse des Zählbytes.

CMOVE (adr1 adr2 n -->)

verschiebt n Byte von Adresse adr1 nach Adresse adr2.

8 FORTH Beispiele auf Heim-Computern

II FORTH Beispiele auf Heim Computern

FORTH ist eine, vom Computer und Prozessor unabhängige Sprache. Alle Beispiele, die bisher gebracht wurden, müssen mit allen FORTH-Versionen realisierbar sein. Sollten einzelne Versionen nicht den vollen Sprachumfang haben, so ist dies nicht weiter tragisch, denn man kann ja die fehlenden Worte unter Verwendung vorhandener Worte neu definieren. Dies ist ein Vorteil von FORTH, daß es keine starre, sondern eine lebendige erweiterbare und vielschichtige Sprache ist.

Wenn jetzt einige Beispiele von FORTH von Heim- oder Personal-computern gebracht werden, so sollen spezifische Eigenschaften dieser Rechner gezeigt werden, die sich besonders gut für die Programmierung in FORTH eignen.

Deshalb wird für den Heimcomputer ATARI 800 die Programmierung von Tönen und Geräuschen betrachtet werden. Dieser Rechner besitzt eine Vielzahl von Möglichkeiten. Die Programmierung in Maschinen-sprache ist langwierig, BASIC ist viel zu langsam. FORTH ist deshalb dafür am besten geeignet.

Das zweite Beispiel wird die Programmierung von Grafik auf dem Apple II sein. Diese Programmteile lassen sich allerdings auch für den ATARI 800 verwenden, da dieser ähnliche Grafikmöglichkeiten wie der APPLE bietet. Auch hier wird sich zeigen, daß der Programmablauf in FORTH wesentlich schneller als in BASIC ist, aber nur einen wesentlich kleineren Anteil an Entwicklungszeit benötigt, wie eine Programmierung in Maschinensprache.

Als drittes Beispiel wird der AIM 65 verwendet, um Steuersignale für Regelungen und ähnliche Anwendungen zu erzeugen. Beim AIM ist FORTH nicht auf Diskette sondern in 2 ROMs untergebracht. Das Wörterbuch des Benutzers wird im RAM angelegt. Auf Besonderheiten dieser Programmierung wird in diesen Abschnitt eingegangen

8.1. SOUND mit dem ATARI 800

Für den ATARI werden zur Zeit zwei FORTH Versionen angeboten. Von dem ATARI PROGRAM EXCHANG (APEX) die Version EXTENDED FIG FORTH und von QUALITY SOFTWARE das QS FORTH. Beide Versionen unterscheiden sich nur wenig. Während APEX FORTH im wesentlichen den Editor, wie er in Kapitel 5.2 beschrieben wurde, verwendet, benutzt QS-FORTH einen Bildschirm orientierten Editor, der die Editiermöglichkeiten des ATARI-Tastenfeldes ausnutzt. Die Dokumentation des ATARI-FORTH beschränkt sich auf die Anführung der verwendeten Definitionen, während QS-FORTH eine bis sehr weit ins Detail reichende Beschreibung der Sprache liefert.

Bei der Erzeugung von Geräuschen benutzen beide Sprachen die gleiche Definition wie in BASIC

n1 n2 n3 n4 SOUND ,

dabei ist n1 die Lautstärke, eine Zahl zwischen 0 und 15, n2 eine Verzerrung des Signals, n3 ist die Frequenz und n4 der Kanal, über den das Signal ausgegeben wird.

In APEX-FORTH wird noch eine Variable FILTER! verwendet. Dies ist die Adresse der Zelle AUDCTL = 53768 Dezimal oder D2 08 HEX. In beiden Versionen muß das Wörterbuch mit den SOUND Definitionen getrennt geladen werden. In QS-FORTH besteht ein Overlay mit dem Editor. Nach der Verwendung von SOUND muß das Wörterbuch des Editors durch Eingabe von EDITOR aufgerufen werden.

Zuerst einige einfache Beispiele:

8 10 121 0 SOUND

Diese Parameterübergabe durch SOUND an den POKEY, den Prozessor, der im ATARI 800 außer anderen Aufgaben auch die Erzeugung von Tönen und Geräuschen übernimmt, erzeugt einen Ton in mittlerer Tonlage, mit mittlerer Lautstärke und ohne Verzerrung auf Kanal 0.

C	60
H	64
A	72
G	81
F	91
E	96
D	108
C	121
H	128
A	144
G	162
F	182
E	193
D	217
C	243

Abbildung 8.1
Frequenztabelle ATARI

Für die Wahl von n2 verwenden wir die in Abbildung 8.1 angegebene Tabelle. Die Angabe von 10 bedeutet die Ausgabe eines unverzerrten Tones. Im nächsten Beispiel werden wird den Ton an- und abschwellen lassen. Dazu haben wir im Textfeld 90 ein Wort WAIT definiert, das eine, gegeben durch die Zahl im TOS variable Zeitverzögerung ermöglicht. Im Textfeld 91 ist durch RR ein langsam anwachsender dann schnell abfallender Ton programmiert. In diesem Beispiel wird durch die DO . . . LOOP Schleife die Tonhöhe verändert.

Im nächsten Wort SIRENE soll die Frequenz verändert werden.

```
SCR # 90
  0   ( ATARI SOUND DEMO )
  1   : 00FF 0 0 0 0 SOUND ;
  2   : 10FF 0 0 0 1 SOUND ;
  3   : 20FF 0 0 0 2 SOUND ;
  4   : 30FF 0 0 0 3 SOUND ;
  5   : OFF 00FF 10FF 20FF 30FF ;
  6   : WAIT 1 DO LOOP ;
  7   : DEEP 1 53768 ! ;
  8
  9
 10
 11
 12
 13
 14
 15

SCR # 91
  0   ( EINFACHEN SOUND WORTE )
  1   FORGET TON
  2   : TON 10 121 0 SOUND ;
  3   : RR 15 0 DO I TON 900 WAIT
  4   LOOP -1 15 DO I TON 400 WAIT -1
  5    +LOOP ;
  6
  7
  8
  9
 10
 11
 12
 13
 14        ;S
 15
```

Abbildung 8.2
Einfache Töne

Die Beispiele im Textfeld 92 zeigen deutlich den Vorteil der FORTH-Sprache. Einzelne Definitionen lassen sich leicht zu neuen Worten zusammenfassen. Der Aufruf eines einzigen Wortes löst dann den ge-

sonten Programmablauf aus.

```
SCR # 92
  0   ( SIRENE )
  1
  2   : RU 200 100 DO 8 10 I 0 SOUND
  3   100 WAIT LOOP ;
  4   : RA 100 200 DO 8 10 I 0 SOUND
  5   100 WAIT -1 +LOOP ;
  6   : SI 10 0 DO RA RU LOOP OFF ;
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

Abbildung 8.3
Sirene

In den Textfeldern 93 bis 97 werden die Tasten 1 bis 8 als Tasten einer Orgel oder eines Klaviers verwendet.

Die Lautstärke wird über die Variable V eingestellt. In (C) wird der Ton C generiert. Für einen klavierähnlichen Klang wird der Grundton, ein davon ein wenig abweichender Ton und der erste Oberton über die Kanäle 0, 1 und 2 ausgegeben. Auf die gleiche Weise werden die übrigen Töne der Tonleiter erzeugt.

Das Wort ?TASTE fragt ab, ob eine der Tasten 1 bis 8 gedrückt wurde. Wenn dies nicht der Fall ist, wird eine –1 auf dem Stapel gelegt.

Um nun zwischen Klavier und Orgel umschalten zu können, werden zwei Wörterbücher angelegt, in welchen die Töne C bis C1 verschieden definiert werden.

Im Wörterbuch PIANO ist es ein abschwellender Ton. In der Schleife

```
: C -1 15 DO I V ! (C) WAIT -1 +LOOP ;
```

wird der Ton C mit abschwellender Lautstärke ausgegeben.
Im Wörterbuch ORGAN ist es ein Ton mit gleichbleibender Lautstärke.

In beiden Wörterbüchern ist ein Verteiler CASE: definiert. So eine Verteileranweisung ist in der Grundversion von FOTH nicht implementiert. Deshalb wurde bisher darauf verzichtet, diese zu beschreiben, da sie in den einzelnen FORTH-Versionen unterschiedlich definiert sind.

Hier wird die CASE-Definition von QS-FORTH verwendet.

CASE: ⟨NAME⟩ ⟨CASEO⟩ ⟨CASE1⟩ . . . ⟨CASEN⟩ ;

⟨NAME⟩ ist der Verteilername. ⟨CASEO⟩ . . . ⟨CASEN⟩ sind die Verteilerworte. Die Zahl auf dem Stapel gibt an, welches Verteilwerwort ausgeführt wird.

Mit 0 wird ⟨CASEO⟩ , mit 1 ⟨CASE1⟩ usw. ausgeführt. Aber Vorsicht ! Wenn eine falsche Zahl eingegeben wird verschwindet das Programm meistens im Urwald. Ein Neustart ist nur durch Aus- und Einschalten des Rechners möglich.

Mit PLAY wird dann gespielt. Mit PIANO PLAY und ORGAN PLAY wird zwischen den beiden Klangfarben umgeschaltet.

Mit PIANO können einzelne Töne auch als Wortfolge eingegeben und gespielt werden. Dies ist mit ORGAN nicht möglich. Das Programm zeigt Abbildung 8.4.

Das Programm STICK AND TRIG zeigt, wie man die Joysticks in den vier USER-Ports abfragen kann. Aus den Zellen 632 bis 635 werden die Werte von den Joystick und aus den Zellen 644 bis 647 werden die Werte von den Triggerknöpfen eingelesen.

```
SCR # 93
  0   ( TON TABELLE ) 90 LOAD
  1   15 VARIABLE V
  2   : (C) V @ 10 243 0 SOUND
  3         V @ 10 240 1 SOUND
  4         V @ 10 121 2 SOUND ;
  5   : (D) V @ 10 217 0 SOUND
  6         V @ 10 214 1 SOUND
  7         V @ 10 108 2 SOUND ;
  8   : (E) V @ 10 193 0 SOUND
  9         V @ 10 190 1 SOUND
 10         V @ 10  96 2 SOUND ;
 11   : (F) V @ 10 182 0 SOUND
 12         V @ 10 179 1 SOUND
 13         V @ 10  91 2 SOUND ;
 14   -->
 15
```

```
SCR # 94
  0   ( TON TABELLE FORTS. )
  1   : (G) V @ 10 162 0 SOUND
  2         V @ 10 160 1 SOUND
  3         V @ 10  81 2 SOUND ;
  4   : (A) V @ 10 144 0 SOUND
  5         V @ 10 142 1 SOUND
  6         V @ 10  72 2 SOUND ;
  7   : (H) V @ 10 128 0 SOUND
  8         V @ 10 126 1 SOUND
  9         V @ 10  64 2 SOUND ;
 10   : (C1) V @ 10 121 0 SOUND
 11          V @ 10 119 1 SOUND
 12          V @ 10  60 2 SOUND ;
 13   : ?TASTE KEY 49 - DUP 0< IF -1
 14      ELSE DUP 7 > IF -1 THEN
 15      THEN ; -->
```

```
SCR # 95
  0  ( PIANO DEFINITIONS )
  1  VOCABULARY PIANO   IMMEDIATE
  2  PIANO DEFINITIONS
  3  : WAIT 100 0 DO LOOP ;
  4  : C -1 15 DO I V ! (C) WAIT
  5    -1 +LOOP ;
  6  : D -1 15 DO I V ! (D) WAIT
  7    -1 +LOOP ;
  8  : E -1 15 DO I V ! (E) WAIT
  9    -1 +LOOP ;
 10  : F -1 15 DO I V ! (F) WAIT
 11    -1 +LOOP ;
 12  : G -1 15 DO I V ! (G) WAIT
 13    -1 +LOOP ;
 14  : A -1 15 DO I V ! (A) WAIT
 15    -1 +LOOP ; -->
```

```
SCR # 96
  0  ( PIANO  FORTS. )
  1  : H -1 15 DO I V ! (H) WAIT
  2    -1 +LOOP ;
  3  : C1 -1 15 DO I V ! (C1) WAIT
  4    -1 +LOOP ;
  5
  6  CASE: TON C D E F G A H C1 ;
  7
  8  : PLAY BEGIN ?TASTE DUP -1
  9         WHILE TON REPEAT
 10         15 V ! ;
 11
 12
 13  -->
 14
 15
```

```
SCR # 97
  0  ( ORGAN DEFINITIONS )
  1  VOCABULARY ORGAN IMMEDIATE
  2  ORGAN DEFINITIONS
  3  15 V !
  4  : C (C) ; : D (D) ; : E (E) ;
  5  : F (F) ; : G (G) ; : A (A) ;
  6  : H (H) ; : C1 (C1) ;
  7
  8
  9  CASE: TON C D E F G A H C1 ;
 10
 11  : PLAY BEGIN ?TASTE DUP -1
 12         WHILE TON REPEAT
 13         OFF ;
 14
 15
```

Abbildung 8.4
ATARI spielt Klavier oder Orgel

```
SCR # 101
  0  ( STICK AND TRIG )
  1  : STICK1 632 C@ ;
  2  : STICK2 633 C@ ;
  3  : STICK3 634 C@ ;
  4  : STICK4 635 C@ ;
  5
  6  CASE: STICK STICK1 STICK2
  7              STICK3 STICK4 ;
  8
  9  : STRIG1 644 C@ ;
 10  : STRIG2 645 C@ ;
 11  : STRIG3 646 C@ ;
 12  : STRIG4 647 C@ ;
 13
 14  CASE: STRIG STRIG1 STRIG2
 15              STRIG3 STRIG4 ; -->
```

```
SCR # 102
  0  ( TESTS )
  1  : TEST1 BEGIN ?TERMINAL 0=
  2                WHILE
  3                4 0 DO I STRIG .
  4                LOOP
  5                CR
  6          REPEAT ;
  7
  8  : TEST2 BEGIN ?TERMINAL 0=
  9                WHILE
 10                4 0 DO I STICK .
 11                LOOP
 12                CR
 13          REPEAT ;
 14  ;S
 15
```

Abbildung 8.5
Joystickabfrage

```
SCR # 100
  0  ( DISPLAY LIST AENDERN ef )
  1  0 VARIABLE DPLA 0 VARIABLE BSPA
  2  560 @ DPLA !   DPLA @ 5 + @
  3  BSPA !
  4  : AUS BSPA @ DPLA @ 5 + ! ;
  5  : DPL1 2000 0 DO I DPLA @ 5 +
  6    LOOP AUS ;
  7
  8  : DPLG DPLA @ DUP 3 + 71 SWAP
  9      C! DUP
 10    6 + >R DUP 18 + R> DO 7 I C!
 11      LOOP DPLA @ DUP 18 + 65
 12      SWAP C! DUP DUP 19 + DPLA @
 13      SWAP C! 20 + SWAP DPLA 256
 14      / C! ;
 15
```

Abbildung 8.6
Display-List

In den beiden Testprogrammen TEST1 und TEST2 werden diese Werte von den Verteiler STRIG oder STICK auf den Bildschirm ausgegeben.

Das Programm in Abbildung 8.6 enthält zwei Worte welche die Display-List ändern. Mit DPL1 wird der Anfang der Bildspeicheradresse in dieser Liste verändert. Der Aufruf dieses Wortes hat ein flackern des Bildschirms und der Anzeige von "Garbadge" zur Folge.

Das Wort DPLG ändert die Display-List zur Anzeige von 12 Zeilen mit 40 Zeichen um. Rückkehr nur mit Reset.
Mit FORTH lassen sich die Grafikmöglichkeiten des ATARI erst richtig ausnutzen. Viele der bekannten Bildschirmspiele sind in FORTH geschrieben.

8.2. FORTH auf APPLE II

Als FORTH-Version für den Apple II wurde hauptsächlich FORTH-79 und für die Grafik SUPER-FORTH verwendet. FORTH-79 enthält den nach den Standard-Richtlinien aufgebauten Wortvorrat.

Der Texteditor ist Bildschirm-orientiert und gibt aber keine Zeilennummern im Textfeld aus. Außerdem ist ein Assembler enthalten. Die Low-Resolution Grafik ist implementiert, dagegen nicht die High-Resolution Grafik.

Die Speicherverteilung von FORTH-79 zeigt Abbildung 8.7.

Für die Low-Resolution Grafik stehen folgende Worte zur Verfügung:

COLOR (n --)

n wählt die Farbe aus.

Die Farben sind als Konstante vereinbart, so daß man auch eine Farbe mit

WHITE COLOR ,
ORANGE COLOR oder
BLACK COLOR aufrufen kann.

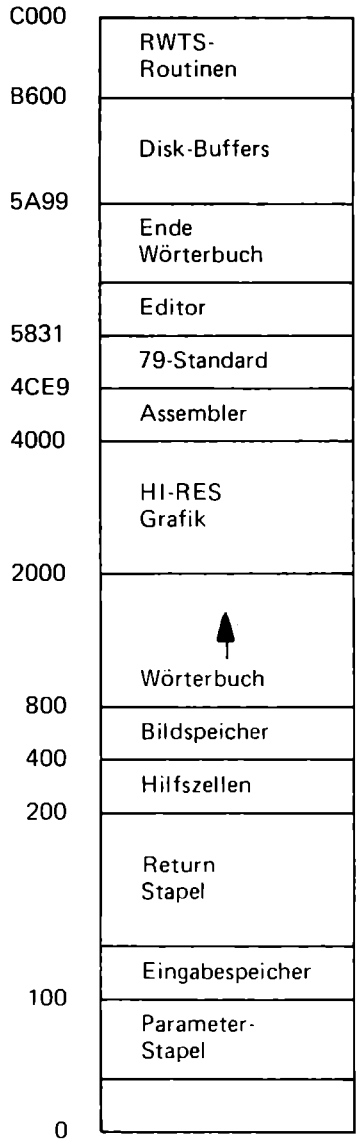


Abbildung 8.7
Speicherverteilung Apple-FORTH-79

Vorher muß allerdings die Grafik aufgerufen werden. Dies geschieht durch das Wort LGR. Will man gemischte Grafik (4-Zeilen Text am

unteren Rand), so muß man die Grafik mit MIXED aufrufen.

Ein Punkt wird in der angewählten Farbe durch PLOT gezeichnet:

PLOT (Y X -->)

Zeichnet Punkt bie X, Y

Die Farbe eines Punktes kann man mit SCRN abfragen.

SCRN (Y X --> n)

n ist die Farbe des Punktes X, Y

Zum Zeichnen von Linien stehen die Worte HLINE und VLINE zur Verfügung.

HLINE (X1 X2 Y -->)

Zeichnet eine horizontale Linie von X1 nach X2, in der Höhe Y.

VLINE (Y1 Y2 X -->)

Zeichnet eine vertikale Linie von Y1 nach Y2 Koordinate X.

Die Rückkehr aus der Grafik in den Textmodus wird durch das Wort TX erreicht.

TX (-->)

Rückkehr in den Textmodus.

In den Abbildungen 8.8 und 8.9 sind nun zwei Beispiele für verschiedene Muster gegeben. Die Farben werden in beiden Beispielen durch den Schleifenparameter I bestimmt. Im zweiten Beispiel wird zu I jeweils noch der Inhalt der Variablen CO addiert, um bei jedem

Schleifendurchlauf eine Farbverschiebung zu erreichen. Für schwarz-weiß Monitore ergibt sich ein Muster mit verschiedenen Helligkeiten

```
SCR# 100
( LOW RES GRAFIK        EF    )  FORGET COL
: COL 8 MOD 2 * COLOR ;
: 2COL 16 SWAP - COL ;
: 1S 19 SWAP DO I COL DUP I SWAP PLOT LOOP ;
: 2S 20 DO I 2COL DUP I SWAP PLOT LOOP ;
: 3S 19 0 DO I I 1S LOOP ;
: 4S 19 0 DO I 39 I - 2S LOOP ;
: 5S 23 2 DO I COL 10 30 I HLINE LOOP ;
: 6S 31 10 DO I COL 2 22 I VLINE LOOP ;
: 7S 2 22 DO I 2COL 10 30 I HLINE -1 +LOOP ;
: 8S 10 30 DO I 2COL 2 22 I VLINE -1 +LOOP ;
: T 20 0 DO 5S 6S 7S 8S LOOP ;
```

Abbildung 8.8
LOW RES GRAFIK

```
SCR# 101
( LOW RES GRAFIK FORTS. ) FORGET CO
0 VARIABLE CO
: #COL CO @ + COLOR ;


: L 15 0 DO I #COL DUP I SWAP PLOT LOOP ;
: LL 15 0 DO 15 I - #COL DUP I 14 + SWAP
   PLOT LOOP ;
: 1Z L LL DROP 1 CO +! ;
: 2Z 20 0 DO I 1Z LOOP ;
: ZZ 20 0 DO 2Z LOOP ;
```

Abbildung 8.9
LOW RES GRAFIK Fortsetzung

Die entgültigen Muster werden durch Aufruf von T bzw. ZZ aufge-

[illegible] Man kann natürlich auch einzelne Schleifen gesondert ablaufen [illegible]

[illegible] die High-Resolution Grafik wurde Super FORTH verwendet. [illegible] verwendeten Befehle sind:

CLEAR (-->)

Löscht den Grafikspeicher

HGR (-->)

Schaltet die Grafik ein

PLOT (X Y -->)

Zeichnen eines Punktes bei X, Y

LINE (X Y -->)

Zeichnet Linie vom letzten PLOT Punkt oder vom Ende der letzten Linie nach X Y

WHITE (-->)

Schaltet Farbe weiß ein

BLACK (-->)

Schaltet Farbe schwarz ein

Will man diese Befehle in einer anderen FORTH-Version implementieren, so kann man folgende Maschinenprogramme benutzen:

```
JSR  $F3D8     HGR
JSR  $F3E2     HGR2    (Grafik mit 4 Zeilen Text)
JSR  $F399     TEXT
```

Die Farbe wird durch den Inhalt der Zelle $E4 bestimmt

E4 : 00 schwarz
E0 : FF weiß

Für den Plotaufruf müssen die Zahlen für die Koordinaten in den dr Registern übergeben werden.

Für PLOT gilt

JSR $F457

mit der X-Koordinate im X-Register (Lower Byte) und Y-Register (Higher Byte) und die Y-Koordinate im Akkumulator.
Der Koordinatenursprung liegt am Bildschirm oben links.

Für LINE gilt

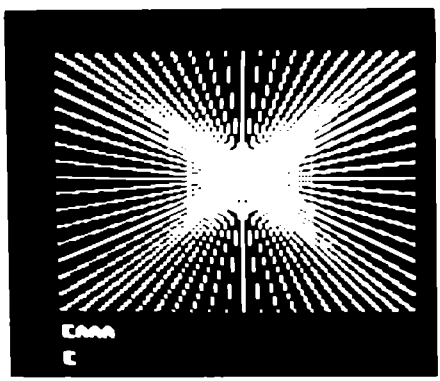
JSR $F538

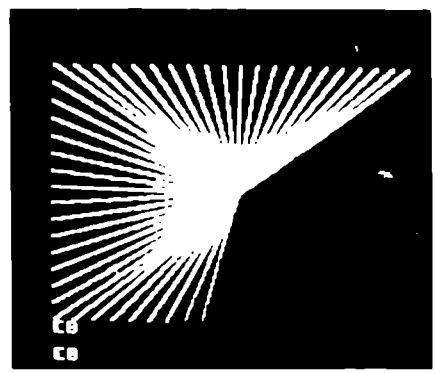
mit der X-Koordinate im A-Register (Lower Byte) und X-Register (Higher Byte) und der Y-Koordinate im Y-Register.

Natürlich muß man bei der Verwendung dieser Routinen sicher sein daß die Speicherbereiche für die Grafik nicht durch FORTH belegt sind.

Nun ein Beispiel:

Das Programm in Abbildung 8.10 erzeugt das strahlenförmige Muster





```
200 VARIABLE X
10 VARIABLE Y
 : C CLEAR HGR WHITE ;
 : S 200 X ! 10 Y ! ;
 : P 100 80 PLOT ;
 : +X X @ 10 + X ! ;
 : -X X @ 10 - X ! ;
 : +Y Y @ 10 + Y ! ;
 : -Y Y @ 10 - Y ! ;
 : L X @ Y @ LINE ;
 : LLL 20 0 DO P L -X   LOOP ;
 : DDD 15 0 DO P L +Y   LOOP ;
 : RRR 20 0 DO P L +X   LOOP ;
 : UUU 15 0 DO P L -Y   LOOP ;
 : B C S LLL DDD RRR UUU ;
```

Abbildung 8.10
Programm zur Erzeugung eines Strahlenkranzes

Dazu werden 2 Variable X und Y eröffnet und ihnen die Werte 200 und 10 zugewiesen. Die erste Definition C enthält die Befehlsfolge CLEAR HGR WHITE. Damit wird der Bildspeicher für die High-Resolution Grafik gelöscht und durch HGR angewählt. Die Farbe, mit welcher gezeichnet wird, ist weiß.

Das Wort S weist den beiden Variablen die Startwerte 200 und 10 zu. Die Wortfolge 200 X ! bedeutet, den Wert 200 nach X speichern. Durch P wird ein Punkt mit der X-Koordinate 100 und der Y-Koordinate 80 angewählt und durch PLOT in der angegebenen Farbe gezeichnet. Dies ist der Mittelpunkt unseres Strahlenkranzes. Die folgenden 4 Definitionen sind Operationen mit den Variablen X und Y. Als Beispiel soll das Wort +X erklärt werden. Mit X G wird der Wert der Variablen X auf den Stapel geholt und durch 10+ um 10 erhöht und danach durch X ! zurückgeschrieben.

Die Definition L ist wiederum ein Grafikbefehl. Von den augenblicklichen Bildkoordinaten wird eine Linie zu den Koordinaten X und Y gezogen. Damit sind die Vorbereitungen für das Zeichen des Strahlenkranzes abgeschlossen.

Die nächste Definition LLL zeichnet in einer Schleife eine Linie vom Mittelpunkt bis zum Wert X, mit Konstanten Y. Bei jedem Schleifendurchlauf wird mit –X, der Wert von X um 10 erniedrigt. Die Strahlen werden am Bildschirm von oben rechts nach oben links gezeichnet.

Dieser Strahlenkranz wird dann mit DDD nach unten, mit RRR nach Rechts und mit UUU nach oben fortgesetzt.
In der letzten Definition ist dann mit B der gesamte Bildaufruf zusammengefasst.

8.3. FORTH auf AIM-65

FORTH für den AIM-65 enthält neben dem FORTH-Kernel noch einen Assembler. Das besondere an dieser FORTH-Version ist, daß sie in ROM (Read Only Memory) geliefert wird. Zwei ROM mit FORTH werden gegen zwei BASIC-ROM ausgetauscht. Abbildung 8.11 zeigt die Speicherverteilung. Für die von FORTH benützten Variablen ist der Speicherbereich $200 bis $260 reserviert. Darüber können vom Benutzer definierte Variable abgelegt werden. Der Eingabespeicher liegt bei $280 bis $2FF. Das Wörterbuch wird ab $300 aufgebaut.

Als erstes Wort ist das Wort TASK schon vereinbart. Das FORTH Kernel mit dem FORTH-Wörterbuch liegt zwischen $B000 und $D000. Wie bei anderen 6502 Systemen liegt der Parameterstapel in der Zero Page und als RETURN-Stapel wird der 6502 Prozessor Stapel verwendet. Für die Eingabe eines FORTH-Programms kann der Text editor des AIM-65 verwendet werden und auch compiliert werden. Das Definitionsende wird mit dem Wort FINIS bezeichnet.

Als kleines Beispiel soll gezeigt werden, wie der 6522 programmiert werden kann. Einzelne Bit des Tores A sollen ein und ausgeschaltet werden können. Die Bit sind mit 1 bis 8 numeriert. Man kann sich vorstellen, daß jedes Bit ein bestimmtes Gerät ein- oder ausschaltet. Vom Tastenfeld soll durch 3 EIN das Gerät Nummer 3 eingeschaltet und durch 3 AUS wieder ausgeschaltet werden. Dabei sollen aber andere Geräte nicht beeinflußt werden. Das Programm zeigt Abbildung 8.12.

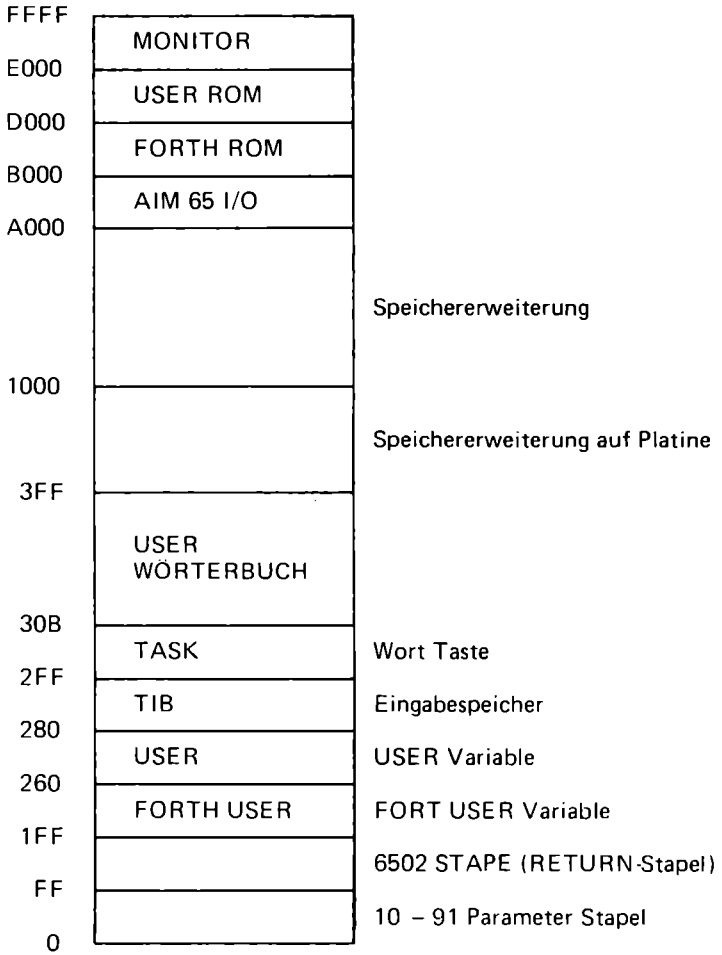


Abbildung 8.11
Speicherverteilung FORTH AIM-65

Die Adresse des Datenrichtungsregisters DDRA und des Tores A TORA wurden als Konstante vereinbart. Das Wort START setzt den Anfangszustand. Es legt eine Null auf dem Stapel ab. Das bedeutet, daß im Anfang alle Bit Null sind. Weiter wird durch Speichern von FF nach DDRA alle Leitungen zu Ausgängen.

Das Wort AN gibt den Wert im TOS an das TOR A aus.

```
HEX
A003 CONSTANT DDRA
A001 CONSTANT TORA
: START 0 FF DDRA C! ;
: AN TORA C! ;
: NEU 01 SWAP DUP 1 = IF DROP ELSE 1
      DO 2 * LOOP THEN ;

: EIN NEU OR DUP AN ;

: AUS NEU FF XOR AND DUP AN ;


 OK
```

Abbildung 8.12
Bitweises Ein- und Ausschalten an Tor A

Im Wort NEU wird festgelegt, welches Bit geschaltet werden soll. Di Nummer des Bits muß vorher auf dem Stapel sein.

Mit 4 NEU wird in der ELSE . . . THEN Schleife eine Eins im TO: durch 2 * drei mal nach links geschoben.

BIT	8	7	6	5	4	3	2	1
VORHER	0	0	0	0	0	0	0	1
NACHHER	0	0	0	0	1	0	0	0

Im Wort NEU wird nun dieses Bitmuster durch die ODER-Funktion mit dem bisherigen Zustand verknüpft. Damit wird Bit 4 zu Ein gesetzt.
Durch 4 AUS wird das Bit 4 wieder ausgeschaltet. Dabei wird durch Exclusive ODER von FF mit dem Bitmuster Bit 4 zu Null, alle anderen Bit zu Eins. Dann kann durch Bilden der UND-Funktion mit dem alten Zustand dieses Bit gelöscht werden. Alle anderen Bitzustände bleiben erhalten.

Der AIM-65 mit FORTH ist ein idealer Computer für Anwendungen in der Prozessteuerung und ähnlichen Anwendungsfällen.

9 Programmbeispiele

9 Programmbeispiele

Die folgenden Beispiele sollen das Verständnis für die Programmierung in FORTH noch vertiefen. Bei allen Beispielen, und das gilt nicht nur in FORTH, muß die Aufgabenstellung und die Algorithmus für die Lösung bekannt sein, um ein Programm lesen zu können. Auch ein PASCAL-Programm ist unleserlich, wenn man nicht weiß, was programmiert wurde. Durch geschickte Namensgebung kann man FORTH-Programme sehr gut leserlich machen. Es folgen zuerst einige Hilfsprogramme.

9.1. Hilfsprogramme

Das folgende Programm ist in den meisten FORTH-Versionen schon enthalten und druckt den Inhalt des Stapels aus, ohne ihn zu zerstören. Nach der üblichen Vereinbarung steht das oberste Element des Stapels am weitesten rechts.

Das Programm in Abbildung 9.1 verwendet die Systemvariable S0 die die Anfangsadresse des Stapels enthält. Mit 'S wird die Adresse des obersten Elementes vor dem Aufruf von 'S auf den Stapel geholt.

Das Wort .BASE gibt den augenblicklichen Wert der Zahlenbasis aus. Bei der Einstellung von Programmen passiert es öfters, daß man ein Programm in der falschen Zahlenbasis compiliert. Die Worte sind dann nicht mehr lauffähig. Deshalb sollte man beim häufigen Umschalten sich vergewissern, in welcher Zahlenbasis man rechnet.

Das nächste Programm in Abbildung 9.2 erzeugt Zufallszahlen (aus L. Brodie Starting FORTH). Vor dem Aufruf des Wortes RNDNR muß eine Zahl auf dem Stapel sein, die die obere Grenze für die Zufallszahl angibt. Die Zahl selbst ist größer oder gleich Null und kleiner als die obere Grenze.

```
SCR # 83
  0   ( STAPELAUSGABE )
  1   ( STAPEL WIRD NICHT VERAENDERT)
  2
  3   : 'S SP@ ;
  4
  5   : TIEF S0 @ 'S - 2 / 1 - ;
  6
  7   : .S CR TIEF IF 'S 2 - S0 @ 2 -
  8        DO I @ . -2 +LOOP
  9        ELSE ." LEER" THEN ;
 10
 11   ( AUSGABE DER ZAHLENBASIS )
 12   : .BASE BASE @ DUP DECIMAL .
 13           BASE ! ;
 14
 15
```

Abbildung 9.1
Ausgabe des Stapels

```
SCR # 89
  0   ( ZUFALLSZAHLEN GENERATOR )
  1
  2   0 VARIABLE RND HERE RND !
  3
  4   : RANDOM RND @ 31421 * 6972 +
  5            DUP RND ! ;
  6
  7   : RNDNR  ( U1 --> U2 )
  8            RANDOM U* SWAP DROP ;
  9
 10   ( 0 <= U2   U1 )
 11
 12
 13
 14
 15
```

Abbildung 9.2
Zufallszahlengerator

Mit

6 RNDNR 1+

werden die Würfelzahlen 1 bis 6 erzeugt. Als Anfangswert für die Generierung einer Zufallszahl wird eine Variable RND verwendet, in die der augenblickliche Wert von HERE gespeichert wird. Bei einem Programm, das immer an der gleichen Stelle steht, erhält man dann immer die gleiche Folge von Zufallszahlen. Besser ist es, wenn man eine Speicherzelle (2 Byte) in RANDOM verwendet, von der man weiß, daß sie sich laufend verändert. Dies kann z. B. eine Timerzelle des Rechners sein.

9.2. Adressenkartei

Das Programm in Abbildung 9.3 zeigt eine Adressenkartei. Dabei werden die Programme Adresseneingabe, Adressenausgabe und virtueller Speicher in abgeänderter Form verwendet. Im ersten Textfeld werden Anfang und Länge des Vornamens, Nachnamens usw. festgelegt.

In

CREATE (STRASSE) 20 , 15 ,

gibt die erste Zahl den Anfang von Strasse innerhalb einer Adresse und die zweite Zahl die maximale Länge an.

```
SCR# 1
( ADRESSEN EINGABE    EF )    : TASK ;
: HILFE 8 LIST ;
: 2@ DUP 2+ @ SWAP @ ;
CREATE (VORNAME) 0 , 10 ,
CREATE (NACHNAME) 10 , 10 ,
CREATE (STRASSE) 20 , 15 ,
CREATE (PLZ) 35 , 4 ,
CREATE (ORT) 39 , 15 ,

: VORNAME (VORNAME) 2@ ;
: NACHNAME (NACHNAME) 2@ ;
: STRASSE (STRASSE) 2@ ;
: PLZ (PLZ) 2@ ;
: ORT (ORT) 2@ ;
```

```
SCR# 2
( ADRESSEN EINGABE  FORTS.   EF )
64 CONSTANT RECLEN

: EIN PAD + SWAP  EXPECT ;
: (EINGABE) HOME CR PAD RECLEN 32 FILL
  ."  VORNAME " VORNAME EIN    CR
  ." NACHNAME " NACHNAME EIN   CR
  ."  STRASSE " STRASSE EIN     CR
  ."      PLZ " PLZ EIN
  ."      ORT " ORT EIN         CR ;


SCR# 3
( SUCHEN       EF )
0 VARIABLE WO  1 VARIABLE #NR
: ANFANG 1 #NR ! ;
: ['] [COMPILE] ' ;
: (VERGL) ( ADR1 ADR2 C --> F )
    BEGIN ROT DUP C@ >R OVER I = R> SWAP
          DUP IF 0
          ELSE DROP >R ROT DUP C@ R>  =
          DUP DUP THEN   WHILE 2DROP 1+
          >R 1+ R> ROT
    REPEAT  >R 2DROP 2DROP R> ;
: LEER PAD RECLEN 32 FILL ;
: WAS ['] 2 - EXECUTE SWAP DROP DUP WO !
  13 WORD COUNT ROT PAD + SWAP
  CMOVE ;


SCR# 4
( ADRESSEN AUSGABE          EF )
: 1L ."  " ;
: 5L ."      " ;
: PRINT ( Z --> ADR )
        PAD + SWAP -TRAILING TYPE ;
: AUS CR 5L 10 0 PRINT 1L 10 10 PRINT CR
      5L 15 20 PRINT CR 4 35 PRINT 1L
      15 39 PRINT CR ;
```

```
SCR# 5
( VIRTUELLER SPEICHER       EF )
 10 CONSTANT START
: INDEX ( N --> ADR )
  RECLEN * 1024 /MOD START + BLOCK + ;
: ANZAHL ( --> N ) 0 INDEX ;
: +NR ( N --> N+1 ) 1 ANZAHL +! UPDATE ;
: !SPEICHER PAD ANZAHL @ INDEX RECLEN CMOVE
  UPDATE +NR ;
: @SPEICHER INDEX PAD RECLEN CMOVE AUS ;
: INIT 1 ANZAHL ! UPDATE ;
: .INHALT CR ANZAHL @ DUP 1 =
          IF ELSE 1 DO I @SPEICHER  CR
          LOOP THEN ;
: #NAME #NR @ @SPEICHER ;
```

```
SCR# 6
( ADRESSEN-KARTEI         EF )
: VERGL PAD WO @ + #NR @ INDEX WO @ +
        32 (VERGL) ;
: FEHLT ." UNBEKANNT" CR ;
: FERTIG SAVE-BUFFERS ;
: (SAVE) ANZAHL @ 8 MOD 0= IF FERTIG THEN ;
: (SUCHE) DO I #NR ! VERGL DUP IF LEAVE
          ELSE DROP THEN LOOP ;
: ENDE ANZAHL @ 1 - #NR @ = ;
: SUCHE LEER WAS 0 ANZAHL @ 1 (SUCHE) IF
       DROP #NAME ELSE ENDE IF FEHLT THEN THEN ;
```

```
SCR# 7
( ADRESSENKARTEI FORTS.     EF )


: EINGABE (EINGABE) !SPEICHER  (SAVE) ;
: LOESCHEN #NR @ INDEX RECLEN 42 FILL ;
: EINTRAG (EINGABE) PAD #NR @ INDEX
          RECLEN CMOVE ;
: .NAME DUP #NR ! @SPEICHER ;
```

```
SCR# 8
( ADRESSENKARTEI HILFSTEXT EF )
( FOLGENDE WORTE SIND VEREINBART:

  EINGABE ( - ) EINGABE DER ADRESSE
  SUCHE <BEGRIFF> <NAME> Z.B.
  SUCHE VORNAME HEINZ
  LOESCHEN ( - ) LOESCHT DEN GERADE GESUCHTEN
  EINTRAG. ERSETZT ALLE BUCHSTABEN DURCH * .
  EINTRAG  ( - ) SETZ EINE EINGABE AN DIE
  GERADE GEFUNDENE STELLE .#NR ENTHAELT DIE
  NUMMER DES EINTRAGS.
  .NAME (N - ) GIBT EINTRAG N AUS.
  .INHALT ( - ) GIBT ALLE EINTRAEGE AUS.
  FERTIG ( - ) SCHREIBT GEAENDERTE BLOECKE
  AUF DISKETTE.                    )

 OK
```

Abbildung 9.3
Adressenkartei

Durch die Eingabe von STRASSE werden diese beiden Zahlen auf den Stapel gelegt.

Im nächsten Textfeld findet dann die Eingabe in den Speicherbereich PAD statt. Im dritten Fextfeld sind Hilfsworte für das Suchen eines Eintrags definiert. Das Wort (VERGL) erwartet vor dem Aufruf drei Angaben auf dem Stapel. adr1 ist die Anfangsadresse einer Zeichenkette, adr2 ist die Anfangsadresse einer zweiten Zeichenkette und c ist ein Begrenzer. Beide Zeichenketten werden miteinander verglichen. Werden zwei ungleiche Zeichen gefunden, so wird dieses Wort mit 0 auf dem Stapel verlassen.

Der Vergleich wird solange fortgesetzt, bis der Begrenzer C in der Zeichenkette bei adr1 gefunden wird. Dann sind beide Zeichenketten bis dahin gleich, und das Wort wird mit 1 auf dem Stapel verlassen.

In einem Programm, das sehr viele Adressen verwalten soll, wird man das Wort in Maschinencode formulieren, um die Suchschnelligkeit zu erhöhen. Auf die Vereinbarung von ['] (Compile-Tick) werden wir

gleich eingehen.

Im Textfeld 4 ist die Ausgabe eines Eintrags aus PAD definiert.
Das Textfeld 5 enthält die Definitionen für den virtuellen Speicher. Dieses Programm ist in FORTH-79 geschrieben. Ein Block ist hier 1k Byte lang. Die Konstante START gibt den Anfangsblock der Adressenkartei und die Konstante RECLEN die Länge eines Eintrags an. Im übrigen sind die Definitionen gleich wie in 6.3, nur daß hier ein Zeichensatz der Länge RECLEN gespeichert wird.

Die nächsten beiden Textfelder enthalten nun die Worte, die der Benutzer für diese Adressenkartei braucht. Durch EINGABE wird ein Eintrag in die Kartei gemacht. Die Eingabe erfolgt nach PAD und wird von dort durch !SPEICHER an das Ende der bisherigen Einträge gemacht.
SUCHE benötigt zwei Folgeangaben.

SUCHE ⟨WAS⟩ ⟨NAME⟩

also z.B.

SUCHE VORNAME HANS

oder

SUCHE PLZ 7500

Für die Eingabe in dieser Form wird nun das Wort

: ['] [COMPILE] ' ;

benötigt.

' ist in FORTH als IMMEDIATE definiert. Wird es als ['] in einer Doppelpunktdefinition aufgeführt, so wird es nicht sofort, sondern bei Aufruf des definierten Wortes ausgeführt. Dabei holt es die Parameterfeldadresse des nächsten Wortes im Eingabespeicher in den Stapel. Bei SUCHE VORNAME also die Adresse von VORNAME, bei SUCHE PLZ die Adresse von PLZ usw.

In dem Wort WAS wird nach |'| aus der Parameterfeldadresse die Codefeldadresse berechnet und ausgeführt. Danach befinden sich die Angaben für den Suchbegriff (Adresse im Eintrag und Länge des Eintrags) auf dem Stapel. Für das Suchen wird nur die Lage innerhalb eines Eintrags benötigt. Diese wird nach WO gespeichert und verbleibt auf dem Stapel. Die folgende Zeichenkette wird nun an diese Stelle nach PAD geschrieben. Vorher wurde PAD mit Leerzeichen gefüllt.

Durch die besondere Definition von (VERGL) braucht man nun für die Suche nicht den vollen Namen sondern nur einen Teil eingeben.

SUCHE NACHNAME MEI

bringt den ersten Eintrag, dessen ersten drei Buchstaben MEI sind.

Im folgenden nun das Benutzerwörterbuch für die Adressenkartei.

EINGABE (-->)

Eingabe eines Eintrags. Dieser wird ans Ende der Liste gehängt.

SUCHE <BEGRIFF> <NAME> (-->)

Sucht den Begriff Name in der Liste. Wird er gefunden, so enthält die Variable #NR die Nummer des Eintrags. Wird er nicht gefunden, so erscheint die Meldung UNBEKANNT.

LOESCHEN (-->)

Löscht den Eintrag desen Nummer in #NR ist. Dabei wird die Zeichenkette durch das Zeichen * ersetzt.

EINTRAG (-->)

Das gleiche wie Eingabe, nur daß die Adresse an die Stelle eingefügt wird, deren Nummer in #NR gespeichert ist.

Durch

SUCHE VORNAME *
EINTRAG

können gelöschte Stellen in der Liste aufgefüllt werden.

.INHALT (-->)

Gibt den gesamten Inhalt der Liste aus.

.NAME (N -->)

Gibt den n-ten Eintrag der Liste aus. Speichert n in #NR.

Das Programm kann noch erweitert werden. Diese Erweiterungen können als Programmierübung angesehen werden. Man kann das Wort

AENDERE <BEGRIFF> <NAME>

einführen, das den Begriff im Eintrag dessen Nummer in #NR enthalten ist in Name umändert. Also

SUCHE STRASSE HOLZWEG
AENDERE STRASSE NEUWEG

Eine andere Erweiterung wäre die Einführung des Wortes WEITER. Damit wird die Suche nach dem Aufinden eines Eintrages fortgesetzt. Dazu wird man den Suchbegriff nicht nach PAD sondern z. B. nach PAD 64 + schreiben, da PAD bei der Ausgabe überschrieben wird.

9.3. Datenbank

Im vorangegangenen Programm Adressliste ist jeweils nur nach einem Begriff in einer Spalte gesucht worden. In dem Programmbeispiel Datenbank kann nun die Kombination mehrerer Begriffe gesucht werden. Als Beispiel nehmen wir eine Datenbank, bei der ein Eintrag aus vier Feldern für jeweils 16 Zeichen besteht. Im ersten Feld wird eine Bezeichnung, in den folgenden drei Feldern die Eigenschaften 1, 2 und 3.

Ein Häuslermarkler kann zum Beispiel in die Bezeichnung eine Kundennummer, und für die Eingenschaften Angaben über Wohnlage, Preis und Zustand machen.

```
SCR# 0
( DATENBANK HILFSTEXT        EF
FOLGENDE WORTE SIND DEFINIERT :
INIT   ANLEGEN EINER NEUEN DATENBANK .
EINGABE  EINGABE IN DIE FELDER .
SUCHE <RET> SUCHEN NACH BEGRIFFEN
LOESCHEN LOESCHEN DES GESUCHTEN EINTRAGS .
EINTRAG NEUER EINTRAG AN DER GESUCHTEN STELLE .
FERTIG SPEICHERN AUF DISKETTE
.NAME ( N) AUSGABE DES EINTRAGS N .
.INHALT AUSGABE DES GESAMTEN INHALTS .
```

```
SCR# 1
( DATENBANK AUFBAU   EF )    : TASK ;
CREATE FELD 4 , 16 ,     : HILFE 0 LIST ;
FELD 2@ DROP CONSTANT L
FELD 2@ SWAP DROP CONSTANT F
: VECTOR CREATE 2 * ALLOT DOES> SWAP 2 * + ;
5 VECTOR #WO
FELD 2@ * CONSTANT RECLEN
: DUP2 DUP DUP ;
: .MSG ." FELD " . ;
: STORE 1 - * PAD + SWAP EXPECT ;

: (EINGABE) HOME CR PAD RECLEN 32 FILL
   FELD 2@  1+ 1 DO I .MSG DUP2 I
   STORE CR LOOP DROP ;
```

```
SCR# 2
( DATENBANK FORTS.     EF )
: LOOK FELD 2@ 1 + 1 DO DUP I 1 - *
       PAD + @ 0= IF 0 ELSE 1 THEN
       I #WO ! LOOP ;
```

```
SCR# 3
( SUCHEN        EF )
: VARIABLE WO  1 VARIABLE #NR
: ANFANG 1 #NR ! ;
: (VERGL) ( ADR1 ADR2 C --> F )
    BEGIN ROT DUP C@ >R OVER I = R> SWAP
          DUP IF 0
          ELSE DROP >R ROT DUP C@ R>  =
          DUP DUP THEN   WHILE 2DROP 1+
          >R 1+ R> ROT
    REPEAT  >R 2DROP 2DROP R> ;

( DATENBANK AUSGABE      EF )
: PRINT F 1+ 1 DO CR I .MSG L DUP
        I 1- * PAD + SWAP TYPE LOOP CR ;

SCR# 5
( VIRTUELLER SPEICHER )
 10 CONSTANT START
: #INDEX ( N --> ADR )
  RECLEN * 1024 /MOD START + BLOCK + ;
: ANZAHL ( --> N ) 0 #INDEX ;
: +NR ( N --> N+1 ) 1 ANZAHL +! UPDATE ;
: !SPEICHER PAD ANZAHL @ #INDEX RECLEN CMOVE
  UPDATE +NR ;
: @SPEICHER #INDEX PAD RECLEN CMOVE PRINT ;
: INIT 1 ANZAHL ! UPDATE ;
: .INHALT CR ANZAHL @ DUP 1 =
          IF ELSE 1 DO I @SPEICHER  CR
          LOOP THEN ;
: #NAME #NR @ @SPEICHER ;
: EINGABE (EINGABE) !SPEICHER ;

SCR# 6
( DATENBANK FORTS.       EF )
: VARIABLE #F
: MOVE> PAD PAD 80 + RECLEN CMOVE ;
: <MOVE PAD 80 + PAD RECLEN CMOVE ;
: AUS MOVE> #NAME <MOVE ;
: LEER PAD RECLEN 32 FILL ;
: #WO-->#F F 1 + 1 DO I #WO @ IF
           I #F ! LEAVE THEN LOOP ;
: #F-->WO #F @ 1 - L * WO ! ;
: SSUCHE LEER (EINGABE) LOOK #WO-->#F
         #F-->WO ANFANG ;
```

```
SCR# 7
( DATENBANK FORTS.        EF )
0 VARIABLE BO
: VERGL PAD WO @ + #NR @ #INDEX WO @ +
        32 (VERGL) ;
: FEHLT ." UNBEKANNT" CR ;
: FERTIG SAVE-BUFFERS ;
: NEXT ( N - N F)  L WO +! 1+ DUP F 1+ = ;
: MUSS ( N - N F)  DUP #WO @ 0= ;
: ((SUCHE)) 1 BO +! #F @  BEGIN NEXT
            IF BO @ IF AUS THEN 0
            ELSE MUSS IF 1 ELSE VERGL THEN THE
            1 =  WHILE REPEAT 0 BO ! #F-->WO ;

SCR# 8
( DATENBANK        ENDE        EF )
: (SAVE) ANZAHL @ 8 MOD 0= IF FERTIG
         THEN ;

: (SUCHE) DO I #NR ! VERGL DUP IF LEAVE
          ELSE DROP THEN LOOP ;
: ENDE ANZAHL @ 1 - #NR @ = ;
: SUCHE LEER WAS 0 ANZAHL @ 1 (SUCHE) IF
      DROP #NAME ELSE ENDE IF FEHLT THEN THEN
: LOESCHEN #NR @ INDEX RECLEN 42 FILL ;
: EINTRAG (EINGABE) PAD #NR @ INDEX
          RECLEN CMOVE ;
: .NAME DUP #NR ! @SPEICHER ;
```

Abbildung 9.4
Datenbank

Für das Suchen von Einträgen kann man nun die einzelnen Angaben kombinieren. Man sucht also z. B. nach der Bezeichnung und der Eigenschaft 2 oder sucht alle Bezeichnungen mit den Eigenschaften 1 und 3 usw.

Im Programm sind für die Eintragungen vier Felder durch CREATE FELD 4 , 16 , , jedes 16 Zeichen lang, festgelegt. Durch Ändern dieser Angabe allein kann das Schema der Datenbank geändert werden. CREATE FELD 8 , 8 , legt 8 Felder mit 8 Zeichen oder CREATE

FIELD 8 , 16 , 8 Felder mit 16 Zeichen. Man sollte aber darauf achten, daß das Produkt aus Feld mal Zeichen die Zahl 64 oder ein Mehrfaches davon ergibt.

Das Programm verwendet im wesentlichen Programmteile, die auch in der Namensliste benötigt wurden, also den virtuellen Speicher und das Suchverfahren. Dieses wurde durch ((SUCHE)) so erweitert, daß, wenn ein Eintrag in einer Spalte gefunden wurde, noch geschaut wird ob in dieser Zeile auch noch der andere Suchbegriff gefunden wird. Wird eine Übereinstimmung mit allen Suchbegriffen gefunden, so wird der Eintrag ausgegeben. Diese Ausgabe geschieht wieder über PAD. Dazu werden die dort gespeicherten Suchbegriffe durch MOVE› weggespeichert und nachher durch ‹MOVE wieder geholt. Vor dem Suchen wird durch LOOK festgestellt, in welchen Spalten überhaupt gesucht werden soll. Diese Angaben werden in dem Feld #WO gespeichert. Die Variable #F enthält die Nummer der Spalte, in welcher als erstes gesucht wird. Alle übrigen Namen sind aus der Adressenkartei bekannt und nochmals in dem Textfeld Hilfe beschrieben.

Beispiele zur Datenbank:

```
.INHALT

FELD 1 EINS        FELD 1 VIER        FELD 1 VIER
FELD 2 ZWEI        FELD 2 EINS        FELD 2 ZWEI
FELD 3 DREI        FELD 3 ZWEI        FELD 3 DREI
FELD 4 VIER        FELD 4 DREI        FELD 4 EINS

FELD 1 ZWEI        FELD 1 DREI        FELD 1 VIER
FELD 2 DREI        FELD 2 BALLO       FELD 2 ZWEI
FELD 3 VIER        FELD 3 EINS        FELD 3 EINS
FELD 4 EINS        FELD 4 GROSS       FELD 4 DREI

FELD 1 DREI        FELD 1 EINS
FELD 2 VIER        FELD 2 BALLO
FELD 3 EINS        FELD 3 GROSS
FELD 4 ZWEI        FELD 4 DREI
```

SUCHE	SUCHE	SUCHE
FELD 1	FELD 1 EINS	FELD 1 VI
FELD 2 HALLO	FELD 2	FELD 2
FELD 3	FELD 3	FELD 3 DR
FELD 4	FELD 4	FELD 4
FELD 1 DREI	FELD 1 EINS	FELD 1 VI
FELD 2 HALLO	FELD 2 ZWEI	FELD 2 ZW
FELD 3 EINS	FELD 3 DREI	FELD 3 DR
FELD 4 GROSS	FELD 4 VIER	FELD 4 FI
FELD 1 EINS	FELD 1 EINS	
FELD 2 HALLO	FELD 2 HALLO	
FELD 3 GROSS	FELD 3 GROSS	
FELD 4 DREI	FELD 4 DREI	

9.4. Lagerverwaltung

Das Programm in Abbildung 9.5 ist ein Beispiel für eine Lagerverwaltung. Eingegeben werden eine 5 stellige Bestellnummer, eine 16 stellige Bezeichnung, der Preis und die Lagermenge.

Für die Programmausführung werden im wesentlichen die gleichen Definitionen wie in den beiden vorangegangenen Beispielen verwendet. Wie bei der Adressliste werden die Eingaben durch CREATE (BEST.NR) usw. festgelegt. Die Eingabe, das Speichern auf Diskette und auch das Suchen sind vollkommen gleich.

```
SCR# 40
( LAGERVERWALTUNG EINGABE                    EF )
: HILFE 48 LIST ;
CREATE (BEST.NR) 0 , 5 ,
CREATE (ARTIKEL) 5 , 16 ,
CREATE (PREIS) 21 , 8 ,
CREATE (MENGE) 29 , 6 ,
 .
: BEST.NR (BEST.NR) 2@ ;
: ARTIKEL (ARTIKEL) 2@ ;
: PREIS (PREIS) 2@ ;
: MENGE (MENGE) 2@ ;
```

```
SCR# 41
( LAGERVERWALTUNG EINGABE FORTS.          EF )
64 CONSTANT RECLEN

: EIN PAD + SWAP  EXPECT ;
: (EINGABE) HOME CR PAD RECLEN 32 FILL
  ."  BEST.NR " BEST.NR EIN    CR
  ."  ARTIKEL " ARTIKEL EIN   CR
  ."    PREIS " PREIS EIN     CR
  ."    MENGE " MENGE EIN     CR ;
```

```
SCR# 42
( SUCHEN                                  EF )
: VARIABLE WO  1 VARIABLE #NR
: ANFANG 1 #NR ! ;
: ['] [COMPILE] ' ;
: (VERGL) ( ADR1 ADR2 C --> F )
    BEGIN ROT DUP C@ >R OVER I = R> SWAP
          DUP IF 0
          ELSE DROP >R ROT DUP C@ R>  =
          DUP DUP THEN   WHILE 2DROP 1+
          >R 1+ R> ROT
    REPEAT  >R 2DROP 2DROP R> ;
: LEER PAD RECLEN 32 FILL ;
: WAS ['] 2 - EXECUTE SWAP DROP DUP WO !
  13 WORD COUNT ROT PAD + SWAP
  CMOVE ;

OK
```

```
SCR# 43
( LAGERVERWALTUNG AUSGABE                 EF )
: 2L ."    " ;
: PRINT ( Z --> ADR )
        PAD + SWAP -TRAILING TYPE ;
: AUS CR  5 0 PRINT 2L 16 5 PRINT 2L
      8 21 PRINT 2L 6 29 PRINT CR ;
```

```
SCR# 44
( VIRTUELLER SPEICHER                                  EF )
 60 CONSTANT START
: #INDEX ( N --> ADR )
  RECLEN * 1024 /MOD START + BLOCK + ;
: ANZAHL ( --> N ) 0 #INDEX ;
: +NR ( N --> N+1 ) 1 ANZAHL +! UPDATE ;
: !SPEICHER PAD ANZAHL @ #INDEX RECLEN CMOVE
  UPDATE +NR ;
: @SPEICHER #INDEX PAD RECLEN CMOVE ;

: INIT 1 ANZAHL ! UPDATE ;
: .INHALT CR ANZAHL @ DUP 1 =
          IF ELSE 1 DO I @SPEICHER  AUS CR
          LOOP THEN ;
: #NAME #NR @ @SPEICHER AUS ;
```

```
SCR# 45
( LAGERVERWALTUNG ZAHLENWANDLUNG                       EF )
: DMUL ( D N - D1)    >R 2DUP 2DUP R> 1 DO D+
       2SWAP 2DUP 2ROT LOOP ;
: DM>PFG 0. PAD 20 + CONVERT 0. 2SWAP CONVERT
    DROP 2SWAP 100 DMUL 2SWAP 2DROP 2SWAP
       2DROP D+ ;
: RECONV <# #S #> PAD 29 + DUP 6 32 FILL
                  SWAP CMOVE ;
: DOCONV 0. PAD 28 + CONVERT DROP ;
: STNEU DUP 0 DOCONV 2SWAP D- RECONV
    PAD #NR @ #INDEX RECLEN CMOVE UPDATE ;
: .DM <# # # 46 HOLD #S #> TYPE ;
: .ZEILE DUP CR 5 .R 2L PAD 5 + 16 TYPE
         DM>PFG ROT DMUL 2L .DM 2DROP DROP ;

 OK
```

```
SCR# 46
( LAGERVERWALTUNG FORTS.                    EF )
: VERGL PAD WO @ + #NR @ #INDEX WO @ +
        32 (VERGL) ;
: FEHLT ." UNBEKANNT" CR ;
: FERTIG SAVE-BUFFERS ;
: (SAVE) ANZAHL @ 8 MOD 0= IF FERTIG THEN ;
: (SUCHE) DO I #NR ! VERGL DUP IF LEAVE
          ELSE DROP THEN LOOP ;
: ENDE ANZAHL @ 1 - #NR @ = ;
: FIN ENDE IF FEHLT THEN ;
: DA 0 ANZAHL @ 1 (SUCHE) IF DROP #NR @ @SPEICHER
    STNEU .ZEILE ELSE FIN THEN ;
: * LEER 0 WO ! 13 WORD COUNT PAD SWAP CMOVE DA ;
: SUCHE LEER WAS 0 ANZAHL @ 1 (SUCHE) IF
      DROP #NAME ELSE FIN THEN ;


SCR# 47
( LAGERVERWALTUNG ENDE                      EF )


: EINGABE (EINGABE) !SPEICHER  (SAVE) ;
: LOESCHEN #NR @ #INDEX RECLEN 42 FILL ;
: EINTRAG (EINGABE) PAD #NR @ #INDEX
          RECLEN CMOVE ;
: .NAME DUP #NR ! @SPEICHER AUS ;




SCR# 48
( LAGERVERWALTUNG HILFSTEXT                 EF )
( FOLGENDE WORTE SIND VEREINBART:

EINGABE      EINGABE ARTIKELS MIT BESTELLNUMMER (5)
             BEZEICHNUNG (16) PREIS (8 MIT DEZIMAL-
             PUNKT UND MENGE (6) .
SUCHE BEST.NR XXXXX
SUCHE ARTIKEL XXXXX ODER XXXLEERZEICHEN
```

```
LAGERBESTANDSAENDERUNG:
<MENGE> * <BESTELLNUMMER> Z.B. 10 * 1002

.INHALT      GIBT LAGERLISTE AUS
.NAME ( N)   GIBT EINTRAG N AUS
EINTRAG ( N) FUEGT NEUEN EINTRAG AN STELLE N EI
FERTIG      SPEICHERN AUF DISKETTE
 OK
```

Abbildung 9.5
Lagerverwaltung

Beispiel zur Lagerverwaltung

```
.INHALT

1000   SCHUHE   29.80   222

1001   SOCKEN   3.95   990

1002   HEMDEN   35.40   90

 OK

12 * 1002
   12   HEMDEN            424.80 OK

.INHALT

1000   SCHUHE   29.80   222

1001   SOCKEN   3.95   990

1002   HEMDEN   35.40   78

 OK
```

Neu in diesem Programm ist die Umwandlung einer Zeichenkette in eine Zahl und umgekehrt, das Rückspeichern einer Zahl als Zeichenkette.

Wird z. B. die Bestellnummer 1005 10 mal verkauft, so wird durch die Eingabe von 10 * 1005 der Gesamtpreis ausgerechnet, die Bestellnummer mit der Bezeichnung, Einzelpreis und Gesamtpreis auf den Bildschirm ausgegeben. Gleichzeitig wird von der Lagermenge die verkaufte Menge abgezogen und die geänderte Eintragung zurückgespeichert.

Für die Umwandlung einer Zeichenkette wird in FORTH-79 das Wort CONVERT verwendet.

```
CONVERT    ( d1 adr1 -- > d2 adr2 )
```

CONVERT wandelt eine Zahl, die als Zeichenkette ab adr1 + 1 gespeichert ist, in eine doppelt lange Zahl d2 um. adr2 ist die Adresse des ersten Zeichens, das keine gültige Ziffer darstellt. Vor dem Aufruf von CONVERT muß eine doppeltlange Zahl d1 auf dem Stapel stehen.

In dem Wort DM>PFG wird der Preis z. B. 29.80 der so als Zeichenkette gespeichert ist, in die Einheit Pfennig als Zahl gewandelt.

Wenn eine Bestellnummer gesucht wurde, so steht sie in PAD. Die Zeichenkette für den Preis beginnt in PAD+21. Durch den ersten Aufruf von CONVERT wird die Zahl vor dem Dezimalpunkt gewandelt. Der zweite Aufruf von CONVERT wandelt dann die Zahl hinter dem Dezimalpunkt um. Durch Multiplikation der ersten Zahl mit 100 und Addition der zweiten Zahl erhält man dann das Ergebnis 2980 als doppeltlange Zahl. Für Preisangaben wird man weiterhin mit doppelt langen Zahlen rechnen, denn damit kann man bis zu Beträgen von 10 Mio DM auf den Pfennig genau rechnen.

In FORTH-Versionen, in denen das Wort CONVERT nicht definiert ist, werden Zahlen, die als Zeichenketten gespeichert sind durch NUMBER gewandelt.

NUMBER (adr --> d)

NUMBER wandelt eine Zeichenkette, die bei adr+1 beginnt und deren Länge in adr gespeichert ist, in eine doppelt lange Zahl d um. Die Adresse eines Dezimalpunktes innerhalb der Zeichenkette wird im DP gespeichert.

Für die Ausgabe wird dann die berechnete Zahl rückgewandelt und auf den Bildschirm ausgegeben. Der veränderte Eintrag wird aus PAD wieder in den Speicher zurück geschrieben.

Schlußbemerkung:

Ich glaube, die vorangegangenen Beispiele zeigen, wie kurz und einfach FORTH-Programme werden können. Sind als Grundstock schon einige Worte vorhanden, so lassen sich leicht neue Programme erstellen oder alte Programme ändern. Wichtig ist allerdings eine gute Programmdokumentation. Auch die Kombination von Programmen ist leicht.

Die beiden Programme Adressliste und Lagerverwaltung compiliert man in zwei verschiedenen Wörterbüchern ADRESSE und LAGER. Zwischen beiden Programmteilen kann man leicht umschalten und so eine Kundenkartei mit Rechnungschreiben, Inventur usw. zusammenbauen.

Aber:
FORTH legt die volle Verantwortung auf den Programmierer.
Alle Fehlermöglichkeiten müssen vom Programmierer erkannt und abgefangen werden. Eine Division durch Null wird nicht erkannt. Diese muß, wenn sie in einem Programm auftreten kann, vom Programmierer abgefangen werden.

In diesem Buch wurde nur auf die Grundstruktur von FORTH eingegangen. Assembler und Erweiterungen für Zeichenketten wurden nicht behandelt. Diese sind meist speziell auf bestimmte Rechner ausgelegt und können in den Handbüchern nachgeschlagen werden.

Ich hoffe aber, daß der Leser Spass an FORTH hat und sich in der eigenwilligen Struktur dieser Sprache zurechtfindet.

ANHANG A

Zusammenstellung der Definitionen

Es folgt nun eine Zusammenstellung der wichtigsten Worte. Diese Liste ist nicht vollständig, da die Sprache nicht starr ist, sondern laufend erweitert und verändert wird. Aber sie enthält die wichtigsten Definitionen, um neue Worte zu schaffen. Außerdem sind in dieser Liste auch nicht die Definitionen enthalten die das FORTH System intern benutzt und auf die ein normaler Benutzer im allgemeinen nicht zugreifen muß.

! (n1 n2 adr --> n1)

Der Wert n2 wird nach adr gespeichert.

(-->)

Wandlung einer Zahlenstelle

#> (--> adr n)

Ende der Formatierung. Auf dem Stapel bleibt die Adresse der Zeichenkette und die Zahl der Zeichen.

#S (-->)

Alle verbleibenden Stellen werden in ASCII-Zeichen gewandelt.

' <NAME> (n1 n2 --> n1 n2 pfa)

Bringt die Parameterfeldaresse von <NAME> in den TOS.

((n1 n2 n3 --> n1 n2 n3)

Kommentarbeginn. Text wird bis) übertragen.

(n1 n2 n3 --> n1 n)

Multiplikation n = n2 * n3

*/ (n1 n2 n3 --> n)

n = n1 * n2 / n3. Das Produkt n1 x n2 wird doppelt lang berechnet

*/MOD (n1 n2 n3 --> n4 n)

n = n1 * n2 / n3 Rest der Division ist n4

+ (n1 n2 n3 --> n1 n)

Addition n = n2 + n3

+! (n1 n2 adr --> n1)

Addiert n2 zum Inhalt einer Variablen, deren Adresse adr auf dem Stapel

+LOOP (n1 n2 n3 --> n1 n2)

Der Zählparameter der Schleife wird um n3 verändert.

, (n1 n2 n3 --> n1 n2)

Speichert n3 in die Speicherzelle auf die HERE zeigt und erhöht den Zeiger um Zwei.

– (n1 n2 n3 --> n1 n)

Subtraktion n = n2 – n3

--> (n1 n2 n3 --> n1 n2 n3)

Text geht im folgenden Textfeld weiter.

–TRAILING (adr n1 --> adr n2)

Ändert für die Ausgabe von Text COUNT so, daß führende Leerzeichen unterdrückt werden.

(n1 n2 n3 --> n1 n2)

Ausgabe von n3 auf Bildschirm.

."XXX" (n --> n)

Ausgabe von XXX als Text.

.R (n1 n2 n3 --> n1 n2)

Tabulator wird auf n3 Stellen gesetzt. Die Zahl wird rechtsbündig ausgegeben

(n1 n2 n3 --> n1 n)

Division n = n2 / n3

/LOOP (u -->)

u Schrittweite.

/MOD (n1 n2 n3 --> n1 n4 n)

n n2 / n3, n4 = Rest.

0< (n1 n2 n3 --> n1 n2 f)

f 1 wenn n3 < 0

0 (n1 n2 n3 --> n1 n2 f)

f 1 wenn n3 = 0

XXX (n --> n)

Beginn der Definition des Wortes XXX.

; (n --> n)

Ende der Wortdefinition.

;S (n1 n2 n3 --> n1 n2 n3)

Ende für Compilirtung

< (n1 n2 n3 --> n1 f)

f = 1 wenn n2 < n3

<BUILDS <NAME> (n --> n)

Trägt <NAME> ins Wörterbuch ein. Belegt Namenfeld, Linkfeld und Codefeld, kein Parameterfeld.

(n1 n2 n3 --> n1 f)

f = 1 wenn n2 = n3

> (n1 n2 n3 --> n1 f)

f = 1 wenn n2 > n3

>R (n1 n2 n3 --> n1 n2)

n3 wird auf den Returnstapel geschrieben.

? (n1 n2 addr --> n1 n2)

X entspricht @ .

?TERMINAL (--> f)

f = 1, wenn irgendeine Taste gedrückt wurde.

@ (n1 n2 adr --> n1 n2 n)

n ist der Inhalt der Speicherzelle adr.

ABS (n1 n2 n3 --> n1 n2 n)

n ist der Absolutwert von n3

ALLOT (n1 n2 n3 --> n1 n2)

Reserviert n3 Byte Speicherplatz im Wörterbuch.

AND (n1 n2 n3 --> n1 n)

n = n2 & n3

BASE (n --> n adr)

Variable, enthält den augenblicklichen Wert der Zahlenbasis.

BLOCK (n --> adr)

Bringt Adresse des Blockes auf Stapel und Block n in Speicher.

C, (b -->)

Ein Byte wird nach HERE gespeichert. HERE wird um Eins erhöht.

C! (n1 b adr --> n1)

Ein Byte wird nach adr gespeichert.

C@ (n1 n2 adr --> n1 n2 b)

Ein Byte wird aus adr geholt.

CFA (Pfa --> cfa)

Rechnet die Parameterfeldadresse in die Codefeldadresse um.

COMPILE

Trägt das folgende Wort während der Laufzeit in das Wörterbuch ein.

CMOVE (adr adr2 n -->)

Verschiebt n Byte von Adresse adr1 nach Adresse adr2.

CONSTANT ⟨NAME⟩ (n1 n2 n3 --> n1 n2)

Definition einer Konstanten ⟨NAME⟩, deren Wert n3 ist.

COPY (n1 n2 n3 --> n1)
Kopiere Textfeld n2 nach n3

CONTEXT (--> adr)

Liefert die Adresse des Wörterbuchs, welches als erstes durchsucht wird.

COUNT (adr --> adr+1 n)

Legt auf dem Stapel die Länge und die Anfangsadresse des Textes ab.
adr ist die Adresse des Zählbytes.

CR (n --> n)

Beginn einer neuen Zeile.

CREATE ⟨NAME⟩

Eintrag von ⟨NAME⟩ ins Wörterbuch. Codefeldadresse ist Parameterfeld-adresse des Eintrags.

CURRENT (. --> adr)

Variable, enthält Adresse des Wörterbuchs, in das die Definitionen eingetragen werden.

D+ (d1 d2 d3 --> d1 d)

d ist die Summe von d1 und d2

D- (d1 d2 d3 --> d1 d)

d ist die Differenz von d2 und d1

D. (d1 d2 d3 --> d1 d2)

Ausgabe von d3

D.R (d1 d2 d3 --> d1)

Druckt d2 in einen d3 breiten Zahlenfeld aus.

D0= (d1 d2 d3 --> d1 d2 f)

f = 1 wenn d = 0

D< (d1 d2 d3 --> d1 f)

f = 1 wenn d3 < d2

D= (d1 d2 d3 --> d1 f)

f = 1, wenn d2 = d3

DECIMAL (n --> n)

Zahlendarstellung Dezimal.

DEFINITIONS <NAME>

Setzt die Adresse von <NAME> in CURRENT Die folgenden Definitionen werden in dieses Wörterbuch eingetragen.

DMAX (d1 d2 d3 --> d1 d)

d = d2 wenn d2 > d3 und d = d3 wenn d3 > d2 ist.

DMIN (d1 d2 d3 --> d1 d)

d = d2 wenn d2 < d3 und d = d3 wenn d3 < d2 ist.

DO (n1 n2 -->)

Eröffnet Schleife von n2 bis n1–1

DOES> (n --> n)

Wird in Definitionsworten benutzt. Beendet den Teil für die COMPILE-Anweisungen und zeigt den Beginn der Laufzeitanweisungen an.

DROP (n1 n2 n3 --> n1 n2)

Oberstes Element löschen

DUP (n1 n2 n3 --> n1 n2 n3 n3)

Duplizieren des obersten Elementes auf dem Stapel.

EMIT (n1 n2 C --> n1 n2)

C wird als 7-bit ASCII-Zeichen auf den Bildschirm ausgegeben.

ERASE (adr n -->)

Füllt ab Adresse adr n Byte mit 00.

EXECUTE (n1 n2 adr --> n1 n2)

Bringt ein Programm, das bei adr anfängt zur Ausführung. adr ist die Codeteiladresse eines FORTH Wortes.

EXPECT (adr n -->)

Eingabe von n Zeichen nach Adresse adr.

FENCE (--> adr)

Variable, enthält die Adresse des ersten ungeschützten Eintrages im Wörterbuch.

FILL (adr n c -->)

Füllt ab Adresse adr n Byte mit ASCII-Zeichen c.

FLUSH (n1 n2 n3 --> n1 n2 n3)

Speichern des augenblicklichen Textfeldes.

FORGET XXX (n --> n)

Löschen der Definition XXX einschließlich aller später gemachten Definitionen

FORTH

Setzt CONTEXT und CURRENT auf FORTH Wörterbuch.

HERE (n1 n2 --> n1 n2 adr)

Bringt die Adresse des nächsten freien Speicherplatzes auf den Stapel.

HEX (n --> n)

Zahlendarstellung Hexadezimal.

HOLD (c -->)

Einfügen des ASCII-Zeichens c in die Zeichenkette.

I (n1 n2 n3 --> n1 n2 n3 n)

Der Laufparameter n der Schleife wird auf den Stapel gelegt.

IMMEDIATE

Erklärt eine Definition als "unmittelbar". Dieses Wort wird sofort ausgeführt.

KEY (n1 n2 n3 --> n1 n2 n3 c)

Warten auf Tasteneingabe.

LATEST (--> adr)

Enthält die Namensfeldadresse der letzten Definition.

LEAVE

Verlassen einer DO . . . LOOP Schleife. Zählparameter bleibt erhalten

LFA (pfa --> lfa)

Rechnet die Parameterfeldadresse in die Verbindungsfeldadresse um.

LIST (n1 n2 n3 --> n1 n2)

Ausgabe von Textfeld n3 auf Bildschirm.

LITERAL (n -->)

Bei Compilierung n vom Stapel in Wortdefinition einfügen.

(--> n)

Bei Laufzeit n auf Stapel ablegen.

LOAD (n1 n2 n3 --> n1 n2)

Compiliere Textfeld n3

M* (n1 n2 --> d1)

d1 = n1 * n2 32-bit Produkt.

M/ (d1 n1 --> n2)

n2 = d1 / n1

MAX (n1 n2 n3 --> n1 n)

n ist die größere der beiden Zahlen n2, n3

MESSAGE (n -->)

Ausgaben der Meldung n

MIN (n1 n2 n3 --> n1 n)

n ist die kleinere der beiden Zahlen n2, n3

n3 wird negativ

MOD (n2 n2 n3 --> n1 n)

n Rest der Division n2 / n3

NFA (pfa --> nfa)

Rechnet die Parameterfeldadresse in die Namenfeldadresse um.

NUMBER (adr --> d)

Wandelt eine Zeichenkette, die bei adr+1 beginnt in eine doppelt lange Zahl.
adr enthält die Länge der Zeichenkette.

OR (n1 n2 n3 --> n1 n)

n = n2 V n3

OVER (n1 n2 n3 --> n1 n2 n3 n2)

2-tes Element des Stapels duplizieren und auf den Stapel legen.

PAD (--> adr)

Adresse des Speichers PAD auf Stapel.

PFA (nfa --> pfa)

Rechnet die Namenfeldadresse in die Parameterfeldadresse um.

QUIT

Beendet Laufzeit ohne Ausgabe von ok.

R (n1 n2 --> n1 n2 n3)

Das oberste Element n3 des Returnstapels wird auf den Rechenstapel kopiert.

R> (n1 n2 --> n1 n2 n3)

n3 wird aus dem Returnstapel zurückgeholt.

ROT (n1 n2 n3 --> n2 n3 n1)

Die obersten 3 Elemente des Stapel zyklisch vertauschen.

SO (--> adr)

Variable, enthält Adresse des Stapelendes

SCR (--> adr)

Variable, enthält Adresse des augenblicklichen Textfeldes.

SIGN (n d --> n)

Gibt innerhalb einer Zahlenformatierung ein Minuszeichen aus.

SP@ (--> adr)

Bringt Adresse von TOS vor dem Aufruf.

SPACE (n --> n)

Ausgabe eines Leerzeichens.

SPACES (n1 n2 n3 --> n1 n2)

Ausgabe von n3 Leerzeichen.

SWAP (n1 n2 n3 --> n1 n3 n2)

Vertauschen der beiden obersten Elemente auf dem Stapel.

TIP (--> adr)

Variable, enthält Adresse des Eingangsspeichers

TYPE (adr n -->)

Ausgabe einer Zeichenkette, deren Anfang bei adr gespeichert ist und die n Zeichen lang ist

U* (u1 u2 --> ud)

ud = u1 * u2

U. (n1 u --> n1)

Ausgabe von u als Zahl ohne Vorzeichen.

U/ (ud u1 --> u2 u3)

ud / u1, u2 Rest, u3 Quotient

U/MOD (ud u1 --> u2 u3)

Dividiert eine 32-bit Zahl durch 16-bit Zal. u3 ist Quotient, u2 der Rest der Division. Alles Zahlen ohne Vorzeichen.

U< (u1 u2 --> f)

Vergleicht die beiden obersten Zahlen in Stapel als Zahlen ohne Vorzeichen.

UPDATE (n1 n2 n3 --> n1 n2 n3)

Kennzeichnung des augenblicklichen Textfeldes als geändert.

USER <NAME> (n -->)

Definitionswort: Trägt eine Variable in einem vom Benutzer festgelegten Bereich ein.

VARIABLE <NAME> (n1 n2 n3 --> n1 n2)

Definition einer Variablen, <NAME>, deren Wert n3 ist.

VOCABULARY <NAME>

Eröffnen eines neuen Wörterbuches.

VLIST (n --> n)

Ausgabe des augenblicklichen Wörterbuches.

WORD (c -->)

Ubernimmt eine Zeichenfolge aus dem Eingabespeicher bis zum Begrenzer c.

XOR (n1 n2 n3 --> n1 n)

n = n2 ∧ n3

[(-->)

Schaltet Compiler aus.

[COMPILE] <NAME>

Ein als IMMEDIATE erklärtes Wort compilieren.

] (-->)

Schaltet Compiler ein.

Kontrollworte:

DO . . . LOOP (n1 n2 n3 --> n1)

Schleife wird von n3 bis n2 in Einerschritten durchlaufen. n2 > n3

+LOOP (n -->)

Erhöhung um n

DO (u1 u2 --> n1 n2) /LOOP (u -->)

u1 obere Grenze, u2 untere Grenze, Erhöhung um u.

IF ⟨WORTE⟩ ENDIF (THEN)
IF (n1 n2 f --> n1 n2)

Die ⟨WORTE⟩ zwischen IF und ENDIF werden nur ausgeführt, wenn f 1 ist

IF ⟨WORTE1⟩ ELSE ⟨WORTE2⟩ ENDIF

IF (n1 n2 f --> n1 n2)

Die ⟨WORTE1⟩ werden ausgeführt, wenn f = 1 ist. Mit f = 0 werden die ⟨WORTE2⟩ ausgeführt.

BEGIN ⟨WORTE⟩ ⟨BEDINGUNG⟩ UNTIL
UNTIL (n1 n2 f --> n1 n2)

Die Ausführung der ⟨WORTE⟩ wird solange wiederholt, bis die ⟨BEDINGUNG⟩ erfüllt, f = 1 ist.

BEGIN ⟨WORTE1⟩ ⟨BEDINGUNG⟩ WHILE ⟨WORTE2⟩ REPEAT ⟨WORTE3⟩
WHILE (n1 n2 f --> n1 n2)

Solange die Bedingung, f = 1, erfüllt ist, wird ⟨WORTE1⟩ und ⟨WORTE2⟩ durchlaufen. Mit f = 0 wird ⟨WORTE2⟩ übersprungen und ⟨WORTE3⟩ ausgeführt.

Definitionswort:

: ⟨NAME⟩ ⟨BUILDS ⟨WORTE1⟩ DOES⟩ ⟨WORTE2⟩ ;

Bei der Compilierung eines Wortes ⟨NAME⟩ werden die ⟨WORTE1⟩ ausgeführt. Bei der Laufzeit dann ⟨WORTE2⟩. Alle mit ⟨NAME⟩ definierten Worte bilden eine Klasse gleicher Eigenschaften.

ANHANG B

Vorschläge zur Namensgebung:
(Nach Unterlagen von FORTH INC.)

Die folgenden Vorschläge sind von FORTH INC. herausgegeben, um eine einheitliche Namensgebung zu erreichen.

Bezeichnung	Bedeutung	Beispiel
!name	Speichern nach name	!SPEICHER
#name	Größe oder Menge	#PUNKTE
'name	1) Adresse von name	'S
	2) Adresse eines Zeigers für name	
(name)	1) In name verwendetes Unterprogramm	(FIND)
	2) Laufzeitverhalten von name	
*name	1) Multiplikation	*ZAHL
	2) Eingabe eines Parameters für eine Scalierung	
+name	1) Addition	+ZAHL
	2) Parametererhöhung	+RUF
	3) Einschalten	
-name	1) Abziehen	–TRAILING
	2) Abschalten	
	3) Umkehrung von name	
	4) Verkehrte Bool'sche Ausgabe (1 = FALSE, 0 = TRUE)	
.name	Druckausgabe von name	.S
/name	1) Division	/ZAHL
	2) Vorzeichenloses aufzählen	/LOOP
	3) "pro"	
1name	1) Erster Name in einer Menge	1SCHALTER
	2) Zahl Eins	
2name	1) Zweiter Name in einer Menge	2SCHALTER
	2) Zahl Zwei	
	3) doppelt lange Operation	2"
:name	Beginn einer Definition	:R
;name	Ende	;CODE

⟨name	1) kleiner als name	⟨
	2) Richtungsangabe	⟨MOVE
⟨name⟩	Bezeichnung für eine Gerätetreiberroutine	⟨TYPE⟩
⟩name	1) nach, wohin	⟩TAPE
	2) Index	
?name	1) Abfrage	?DUMP
	2) Abfrage einer Bedingung mit Abbruch	?STACK
	3) Ausgabe des Inhalts von name	?N
@name	Aus name holen	@N
Cname	Ein Byte lang	C,
Dname	Doppelt lang (4 Byte)	D+
Mname	Operator für gemischte Rechenoperationen	M*
Tname	6 Byte lang	T*
Uname	Vorzeichenlos	U–
|name|	Ausführung während der Compilierung	|'|
\name	Vorzeichenloses runterzählen	\LOOP
name!	Speichere nach name	SP!
name"	Zeichenkette folgt	ABORT" xxx"
name,	Ins Wörterbuch speichern	C,
name⟩	1) Abschluß	#⟩
	2) Richtung	MOVE⟩
name?	Abfrage einer Bedingung mit Rückgabe eines Boolschen Wertes	DONE?
name@	Von name holen	SP@

ANHANG C

Vorschläge zum Programmeintrag in ein Textfeld
(nach Unterlagen von FORTH INC.)

Abweichend von fig-FORTH macht FORTH INC. folgende Vorschläge für das Schreiben eines Programms.

1. Stapelveränderungen:

In jeder Doppelpunkt oder Codedefinition, welche den Stapel ändert, muß diese in einem Kommentar angegeben werden. Das Format des Kommentars ist

(Eingabe – Ausgabe)

wobei das am weitesten rechts stehende Element im TOS ist.

Beispiele:

: TYPE (a n) nur Eingabe
: –FOUND (– a a't) nur Ausgabe
: "CODE (a – n) Ein- und Ausgabe

Folgende Abkürzungen werden verwendet:

a	16-bit Adresse
b	8-bit Byte
c	ASCII-Zeichen
n	16-bit Zahl mit Vorzeichen
u	16-bit Zahl ohne Vorzeichen
d	32-bit Zahl mit Vorzeichen
ud	32-bit Zahl ohne Vorzeichen
l c	Position auf dem Bildschirm l Zeilen c Spalten
s d n	Anfang, wohin, Anzahl (Save, destination, count)
y x	Koordination für Bildschirmgrafik
f l	Erster , letzter (einschließlich)
f l+1	Erster , letzter (ausschließlich)

Werden mehrere Parameter der gleichen Art verwendet, werden sie durch Apostroph oder durch einen Index unterschieden.

(n a a' -- n1 n2)

2. Allgemeine Kommentare

Alle Textfelder müssen in der Zeile Null einen Kommentar enthalten, der die folgenden Wortdefinitionen genügend beschreibt.

Gutes Beispiel: (Gemischte Arithmetik)
Zu wortreich: (Das Textfeld enthält einige doppelt lange Definitionen)
nutzlos: (MISC.OPS)

Außerdem sollte die Kommentarzeile eine Kurzbezeichnung des Programmierers und das Datum enthalten.

Kommentar im Textfeld, der nicht den Stapel betrifft sollte kurz und nur, wenn unbedingt nötig, angegeben werden.

Gutes Beispiel 40961 (TORA)
zuviel DUP 0 = IF (bircht ab wenn gleich Null)

3. Leerzeichen und Leerzeilen

Leerzeilen im Textfeld sind nützlich. Sie sollen dazu verwendet werden verschiedenartige Definitionen oder Blöcke von Definitionen zu trennen.

Die erste Zeile leer zu lassen ist nutzlos, ebenso außer am Ende zwei Leerzeilen. Zeile 15 sollte leergelassen werden.

Anmerkung:
FORTH-79 von FORTH INC. verwendet für Kommentare sogenannte Shadow-blocks. Zu jedem Textfeld mit Wortdefinitionen gehört ein zweites Textfeld (der Schatten) der Kommentar enthält. Zwischen beiden Textfeldern kann durch Tastendruck umgeschaltet werden.

Definitionen sollten am linken Rande, mit einem Leerzeichen nach der Zeilennummer beginnen. In einer Zeile sollte keine zweite Definition beginnen, es sei den, es handelt sich um gleichartige kurze Definitionen. Nach dem Namen ein Leerzeichen bis zum Kommentaranfang und drei Leerzeichen bis zum Definitionsbeginn.

Folgende Zeilen der gleichen Definition sollten immer um drei Leerzeichen mehr als in der vorangegangenen Zeile einrücken. Durch weiteres Einrücken kann eine geschachtelte Struktur sichtbar gemacht werden.

4. Hilfe

Jedes Anwenderprogramm sollte ein Textfeld enthalten das mti HILFE (HELP) aufgerufen werden kann. Dieses Textfeld sollte die nötigsten Worte zur Programmbedienung enthalten, damit der Benutzer im Notfall Hilfe schreien kann.

ANHANG D

Spiel-Programm in PHS-FORTH für CBM-Computer

Crazy Runner

Die Programmiersprache FORTH erlebte in den letzten Monaten einen nicht unerheblichen Aufschwung in Deutschland. Inzwischen sind mehrere FORTH-Compiler, deren Herkunft in fast allen Fällen die FORTH Interest Group ist, auf dem Markt erhältlich.

Das folgende FORTH-Programm wurde mit einem Compiler der Firma phs/SLS Hannover erstellt, der im Gegensatz zu anderen Compilern den Vorzug hat, mit einem Editor ausgestattet zu sein, der keine wesentlichen Einschränkungen zum gewohnten Basic-Editor hat.

Obwohl der Compiler einen Macroassembler enthält, wurde im folgenden Programm kein Gebrauch davon gemacht. phs-FORTH ist schnell genug, um die elf Objekte "gleichzeitig" in fließender Bewegung auf dem Monitor darzustellen. Es sind sogar zwei Verzögerungsschleifen notwendig, um dem Spieler überhaupt noch eine Chance einzuräumen.

Ziel des Spieles ist es, den weißen Punkt von der linken Bildschirmseite über 10 Bahnen auf die Rechte zu befördern. Auf jeder Bahn läuft jedoch ein kleiner grauer Block auf und ab, mit dem man nicht zusammenstoßen darf. Geschieht dieses trotzdem, so wird der Block weiß wenn er gerammt wurde. Der Punkt verschwindet, falls er einem Block im Wege ist. Hat man das Ziel erreicht, so wird die Schwierigkeitsstufe erhöht. Ab Stufe 6 schließlich ist es fast unmöglich ans Ziel zu gelangen. Die Bewegung erfolgt wie gewohnt mit dem Zehnerblock der Tastatur.

Doch nun zum Programm:
Zunächst wird das Wort Bild definiert, welches später die Graphik auf den Bildschirm bringt.

Die nächste Definition heißt INKEY und wartet nach Aufruf auf einen Tastendruck. Nun folgt die Deklaration von Konstanten und Variablen: PUNKT steht für das Bildschirmsymbol des Spielers. Analog dazu ist BLOCK das Symbol der wandernden grauen Felder.

ANFANG und ENDE sind die Bildschirmadressen des Starts und Ziels. POS. und RICHT. markieren die Anfänge der Tabellen für die Positionen und Richtungen der einzelnen Blöcke.

Die Variable LEVEL gibt über den Schwierigkeitsgrad Auskunft, während RI, FLAG, P und J Hilfsvariablen sind.

POSITION ist eine Variable, die die Bildschirmposition des Punktes enthält. Nun erfolgt wieder eine Definition (NEU), die die oben erwähnten Tabellen initialisiert. Das Wort GEMAELDE führt zunächst BILD aus und zeichnet dann die 10 Blöcke auf das Bild. Die Routine TASTE fragt die Tastatur ab und wandelt die Eingabe sofort in das Positionsincrement (1,–1,40,–40) um. Vor dem Gleichheitszeichen steht jeweils der Tastencode, der je nach Tastatur des Rechners einer Änderung bedarf. Mit dem folgenden kleinen Programm läßt sich der Tastencode leicht ermitteln:

```
: TAST BEGIN 151 C' . BREAK AGAIN ;
```

Doch weiter im Programm: als nächstes folgt BEWEGEPUNKT. Wie der Name schon sagt ist dieser Befehl für die Bewegung des Punktes verantwortlich. Das Wort BEWEGEBLOCK bewegt natürlich die 10 Blöcke. Am Ende dieses Befehls finden wir auch die schon erwähnten Warteschleifen, die mittels DO LOOP realisiert wurden. Danach steht die Routine SUB, die genau solange das Spiel ermöglicht, bis der Spieler das Ziel erreicht hat oder scheitert. Der Befehl SPIEL faßt alle Definitionen zum eigentlichen Programm zusammen und gibt noch begleitende Kommentare aus.

Das Programm läßt sich natürlich genauso im Editor wie auch im Direktmodus eintippen. Ausgeführt wird es einfach durch die Eingabe von SPIEL.

```
: BILD ." ▩"
." ▩                    ┌─┬─┬─┬
```

```
NEU
-32266 POS.       !
-32558 POS.  2 + !
-32540 POS.  4 + !
-32430 POS.  6 + !
-32424 POS.  8 + !
-32356 POS. 10 + !
-32228 POS. 12 + !
-32114 POS. 14 + !
-31912 POS. 16 + !
-31902 POS. 18 + !
 40 RICHT.       !
 40 RICHT.  2 + !
 40 RICHT.  4 + !
 40 RICHT.  6 + !
-40 RICHT.  8 + !
 40 RICHT. 10 + !
-40 RICHT. 12 + !
-40 RICHT. 14 + !
-40 RICHT. 16 + !
-40 RICHT. 18 + !

GEMAELDE
BILD ." 3"
18 0
DO
  BLOCK POS. I + @ C!
  2
+LOOP
ANFANG POSITION !

TASTE
151 C@
DUP 41 = IF DROP 1 ;S ENDIF
DUP 50 = IF DROP -40 ;S ENDIF
DUP 42 = IF DROP -1 ;S ENDIF
    18 = IF 40 ;S ENDIF
0
```

```
: BEWEGEPUNKT
  TASTE RI !
  POSITION @ RI @ + DUP C@
  DUP 32 = SWAP PUNKT = OR
  IF
    32 POSITION @ C!
    RI @ POSITION +!
    PUNKT POSITION @ C! DROP
  ELSE
    DUP C@ 102 =
    IF 1 FLAG ! 160 SWAP C!
    ELSE DROP 0 FLAG !
    ENDIF
  ENDIF
POSITION @ ENDE = IF 500 RI !
                   ENDIF
;
: BEWEGEBLOCK
  LEVEL @ 0
  0 FLAG !
  DO
    59464 C@ 10 MOD
    2 * DUP J !
    POS. + @
    RICHT. J @ + @
    + DUP P !
    C@ 32 =
    IF
      32 POS. J @ + @ C!
      P @ POS. J @ + !
      BLOCK P @ C!
    ELSE
      P @ C@ PUNKT <>
      IF
        RICHT. J @ + @ -1 *
        RICHT. J @ + !
      ELSE
        1 FLAG !
        32 POS. J @ + @ C!
           POS. J @ + @ RICHT. J @ + C!
      ENDIF
    ENDIF
  LOOP
```

```
 LEVEL @ 5 <=
 IF
   6 LEVEL @ - 100 * 0
   DO
   LOOP
 ELSE
   40 0 DO LOOP
 ENDIF

 SUB NEU GEMAELDE
 ." ▒                              LEVEL:" LEVEL @ .
0 FLAG !
BEGIN
  BREAK
  BEWEGEBLOCK
  FLAG @ IF ;S ENDIF
  BEWEGEPUNKT
  FLAG @
  RI @ 500 = OR IF ;S ENDIF
AGAIN

 SPIEL
 1 LEVEL !
 BEGIN
   SUB
   FLAG @
   IF
     ." ▒DU HAST VERLOREN"
   ELSE
     1 LEVEL +!
     ." ▒DU HAST ES GESCHAFFT !!"
   ENDIF
   0 158 C!
   1000 0 DO LOOP
   INKEY
 AGAIN
```

Und nun viel Spaß mit Crazy Runner.

Bezeichnungen:

n, n1	16 bit Zahlen mit Vorzeichen
d, d1	32 bit Zahlen mit Vorzeichen
u, u1	16 bit Zahl ohne Vorzeichen
addr	Adresse
b	8 bit Byte
c	7 bit ASCII-Zeichen
f	Bool'sche Zahl

Bei der Angabe der Veränderungen des Stapels ist links der Zustand des Stapels vor dem Aufruf des Wortes, rechts der Zustand des Stapels nach dem Aufruf des Wortes angegeben. Das in dieser Bezeichnung am weitesten rechts stehende Element ist oberstes Element des Stapels.

STAPELÄNDERUNGEN:

DUP	(n --> n n)	Verdoppeln der obersten Zahl
DROP	(n -->)	Löschen der obersten Zahl
SWAP	(n1 n2 --> n2 n1)	Vertauschen der beiden obersten Zahlen
OVER	(n1 n2 --> n1 n2 n1)	Zweite Zahl des Stapels im TOS kopieren
ROT	(n1 n2 n3 --> n2 n3 n1)	Zyklisches Vertauschen der obersten drei Zahlen
–DUP	(n --> n ?)	Verdoppeln der Zahl, nur wenn sie nicht Null ist
>R	(n --> n ?)	Oberste Zahl auf den Return-Stapel schreiben
R>	(--> n)	Oberste Zahl des Return-Stapels auf den Stapel holen
R	(--> n)	Oberste Zahl des Return-Stapels auf den Stapel kopieren

ARITHMETISCHE UND LOGISCHE BEFEHLE:

+	(n1 n2 --> sum)	Addieren
D+	(d1 d2 --> sum)	Addieren doppelt langer Zahlen
–	(n1 n2 --> diff)	Subtrahieren (n1 – n2)
*	(n1 n2 --> prod)	Multiplizieren
/	(n1 n2 --> quot)	Dividieren (n1 / n2)
MOD	(n1 n2 --> rem)	Rest der Division (Modulo)
/MOD	(n1 n2 --> rem quot)	Quotient und Rest der Division
*/MOD	(n1 n2 n3 --> rem quot)	Berechnung von n1 * n2 / n3 mit doppelt langen Zwischenergebnis
*/	(n1 n2 n3 --> quot)	Wie */MOD, nur Quotient auf dem Stapel
MAX	(n1 n2 --> max)	Max ist die größere der beiden Zahlen n1, n2
MIN	(n1 n2 --> min)	Min ist die kleiner der beiden Zahlen n1, n2
ABS	(n --> absolute)	Absolutwert einer 16 bit Zahl
DABS	(d --> absolute)	Absolutwert einer 32 bit Zahl
MINUS	(n --> –n)	16 bit Zahl wird negativ
DMINUS	(d --> –d)	32 bit Zahl wird negativ
[illegible]	[illegible]	[illegible]

DO	(end+1 start -->)	Setzt Schleifenparameter
LOOP	(-->)	Erhöht Zählparameter um Eins
I	(--> n)	Holt Zählparameter auf Stapel
LEAVE	(-->)	Verlässt die Schleife beim nächsten LOOP oder +LOOP
DO . . . +LOOP		Wie DO . . . LOOP, aber addiert n zu Zählparameter
+LOOP	(n -->)	
IF . . . ENDIF	(f -->)	Programm zwischen IF und ENDIF (THEN) wird ausgeführt,
IF . . . THEN	(f -->)	wenn der Wert im TOS ungleich Null ist
IF . . . ELSE . . . ENDIF	(f -->)	Programm zwischen IF und ELSE wird ausgeführt, wenn Wert
IF . . . ELSE . . . THEN		im TOS ungleich Null ist, sonst wird Programm zwischen ELSE und ENDIF (THEN) ausgeführt.
BEGIN . . . UNTIL	(f -->)	Programm zwischen BEGIN und UNTIL (END) wird solange aus-
BEGIN . . . END	(f -->)	geführt, bis im RPS vor UNTIL (END) ein Wort ungleich Null ist
BEGIN . . . WHILE		Solange der TOS vor WHILE ungleich Null ist, wird Programm
. . . REPEAT	(f -->)	zwischen BEGIN und REPEAT ausgeführt. Mit TOS gleich Null vor WHILE wird Schleife verlassen.

SPEICHERVERÄNDERUNG:

@	(addr --> n)	Holen einer Zahl von Adresse addr
!	(n addr -->)	Speichern einer Zahl nach Adresse addr
C@	(addr --> b)	Holen eines Byte von Adresse addr
C!	(b addr -->)	Speichern eines Byte nach Adresse addr
?	(addr -->)	Inhalt einer Variablen ausdrucken
+!	(n addr -->)	Wert einer Variablen um n erhöhen
CMOVE	(from to u -->)	u Byte im Speicher verschieben
FILL	(addr u b -->)	u Byte ab Adresse addr mit b füllen
ERASE	(addr u -->)	u Byte ab Adresse addr mit 00 füllen
BLANKS	(addr u -->)	u Byte ab Adresse addr mit Leerzeichen füllen

VERGLEICHSBEFEHLE:

<	(n1 n2 --> f)	f = 1, wenn n1 < n2
>	(n1 n2 --> f)	f = 1, wenn n1 > n2
=	(n1 n2 --> f)	f = 1, wenn n1 = n2
0<	(n --> f)	f = 1, wenn n kleiner Null
0=	(n --> f)	f = 1, wenn n gleich Null
U<	(Z1 u2 --> f)	f = 1, wenn U1 und U2.
U<	(u1 u2 --> f)	f = 1, wenn u1 < u2

ZAHLENSYSTEME:

DECIMAL	(-->)	Dezimalrechnung
HEX	(-->)	Hexadezimalrechnung
BASE	(--> addr)	Variable, enthält Zahlenbasis

DEFINITIONSWORTE:

: xxx	(-->)	Beginn der Definition von xxx
;	(-->)	Ende der Definition
VARIABLE xxx	(n -->)	Definition einer Variablen xxx mit Wert n
	xxx: (--> addr)	
CONSTANT xxx	(n -->)	Definition einer Konstanten xxx mit Wert n
	xxx: (--> addr)	
<BUILDS . . . DOES>	does: (--> addr)	Definiert neue Compiler Worte mit unterschiedlichen Compile und Laufzeitverhalten
CREATE xxx	(-->)	Trägt xxx ins Wörterbuch ein, mit HERE als Codefeldaresse. Bit 8 der Namenlänge gleich Eins
SMUDGE		Ändert Bit 8 der Namenlänge in obersten Eintrag

EIN- UND AUSGABE:

.	(n -->)	Druckt Wort des TOS aus
R	(n fleidwidth -->)	Druckt rechtsbündig (Feldbreite im TOS) aus
D.	(d -->)	Druckt doppelt lange Zahlen aus
D.R	(d fieldwidth -->)	Druckt doppelt lange Zahlen rechtsbündung aus
U.	(u -->)	Druckt TOS als Zahl ohne Vorzeichen aus
CR	(-->)	Gibt ein Carriage Return aus
SPACE	(-->)	Gibt ein Leerzeichen aus
SPACES	(n -->)	Gibt ein Leerzeichen aus
." xxx "	(-->)	Gibt Text xxx aus
TYPE	(addr u -->)	Gibt u Zeichen, gespeichert ab Adresse addr aus
COUNT	(addr --> addr+1 u)	Ändert Längenbyte für TYPE Ausgabe
?TERIMINAL	(--> f)	f = 1, wenn eine Taste gedrückt
KEY	(--> f)	Eingabe von Zeichen vom Tastenfeld
EMIT	(c -->)	Ausgabe des ASCII-Zeichen C
EXPECT	(addr n -->)	Erwartet n Byte Eingabe; speichert ab Adresse addr
WORD	(c -->)	Liest ein Wort, begrenzt durch c, aus Eingabespeicher

EIN- AUSGABEFORMATIERUNG:

NUMBER	(addr --> d)	Setzt eine Zeichenkette ab Adresse addr in eine doppelt lange Zahl um

SIGN	(n d --) d)	Setze Vorzeichen ein
#)	(d --) addr u)	Beende Formatierung
HOLD	(c --))	Füge ASCII-Zeichen c in Zeichenkette ein

WÖRTERBUCH:

CONTEXT	(--) addr)	Bringt Adresse des CONTEXT Wörterbuches
CURRENT	(--) addr)	Bringt Adresse des CURRENT Wörterbuches
FORTH	(--))	Setzt CONTEXT und CURRENT auf FORTH Wörterbuch
DEFINITIONS	(--))	Setzt CURRENT auf CONTEXT
VOCABULARY xxx	(--))	Erzeugt Wörterbuch xxx
VLIST	(--))	Listet Inhalt des Wörterbuches
FENCE	(--))	Systemvariable, enthält die Adresse des obersten geschützen Eintrags im Wörterbuch

SYSTEM WORTE:

(	(--))	Beginn eines Kommentars, Kommantierende ist)
FORGET xxx	(--))	Löschen eines und aller späteren Einträge im Wörterbuch
ABORT	(--))	Fehlerabbruch
'xxx	(--) addr)	Suche Codefeldadresse von xxx
HERE	(--) addr)	Adresse der nächsten freien Stelle im Wörterbuch
PAD	(--) addr)	Adresse von PAD
IN	(--) n)	Offset zum Eingabespeicher
SP@	(--) addr)	Adresse des TOS
ALLOT	(n --))	Speicher von n an Adresse HERE
,	(n --))	
C,	(b --))	Speicher des Byte b an Adresse HERE

DISKETTENSPEICHERUNG:

LIST	(n --))	Textfeld (Screen) n wird ausgegeben
LOAD	(n --))	Textfeld (Screen) n wird compiliert
BLOCK	(n --) addr)	Diskettenblock n wird eingelesen, Anfangsadresse im Speicher auf Stapel
FLUSH	(--))	Geänderte Blöcke werden auf Diskette geschrieben.
UPDATE	(--))	Augenblicklichen Block als geändert markieren
EMPTY-BUFFERS	(--))	Alle Blöcke im Speicher löschen
BLK	(--) addr)	Variable, enthält augenblickliche Blocknummer
B/BUF	(--) n)	Constante, Blockgröße in Byte
SCR	(--) addr)	Variable, enthält augenblickliche Textfeldnummer

© COPYRIGTH by Ing. W. Hofacker GmbH, 1982

Hofacker-Bücher Deutsch

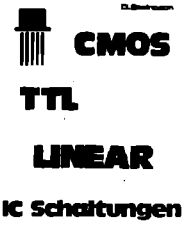
TBB-Transistor-, Berechnungs- und Bauleitungs-HB, Band 1, Hofacker
Völlig neu überarbeitete Auflage. Das Buch soll bei der täglichen Arbeit im Labor, in der Werkstatt oder am Elektronik-Hobbytisch Ihnen ein guter Begleiter sein. Berechnungsgrundlagen, Berechnungsbeispiele, Tabellen, Vergleichslisten, Digitaltechnik, Netzgeräte, BASIC-Programme zur Berechnung spezifischer Schaltungen usw. sind in übersichtlicher Form dargestellt. Ca. 300 Seiten.
Best.-Nr. 1 **29,80 DM**

Electronic im Auto, H. Gebauer
Mit Handbuch für Polizeiradar. Ein Buch für jeden technisch interessierten Autofahrer. Es zeigt Ihnen die vielen Möglichkeiten zur Verbesserung von Sicherheit, Leistung und Fahrkomfort in Ihrem Auto. Thyristorzündung, Beschleunigungsmesser, Drehzahlmesser, Batterieladegerät, Alarmanlagen u. v. a.
Best.-Nr. 3 **9,80 DM**

IC-Datenbuch, D. Steinbach
Daten- und Auswahllisten der gebräuchlichsten integrierten Schaltkreise. Digital und analog. Gerade bei ICs ist es wichtig die Anschlußfolgen genau zu kennen. Die wichtigsten TTL-Schaltkreise, NF-Verstärker, C-Mos Serie, lineare Schaltungen wie Operationsverstärker, Komparatoren, Spannungsregler, Trigger-Schaltungen, u. v. a. Das IC-Datenbuch wird auch Ihnen ein unentbehrlicher Begleiter bei allen Arbeiten mit integrierten Schaltungen sein.
Best.-Nr. 5 **9,80 DM**

TBB-Handbuch, Band 2, Hofacker
Dieses Buch ist die Fortsetzung des erfolgreichen Handbuches, Band 1. Ein Buch, das sich in der Hand des Praktikers bestens bewährt hat. Weitere neueste Schaltbeispiele und Berechnungsgrundl., Experimentier- und Versuchsbeschreibungen. Integrierte Spannungsregler, Wärmeableitung, Operationsverstärker Einführung, RC-Zeitglieder, Transistortester u. v. a.
Best.-Nr. 2 **19,80 DM**

IC-Handbuch, C. Lorenz
Ein Handbuch für digitale und lineare integrierte Schaltungen. Daten- und Auswahl-, Vergleichslisten, Gehäuseformen, Grundlagen, viele Schaltbeispiele, Printvorlagen, u. v. a. Alles über TTL-Technik, C MOS, MOS-Schaltungen, integrierte NF-Verstärker u. v. a.
Best.-Nr. 4 **19,80 DM**



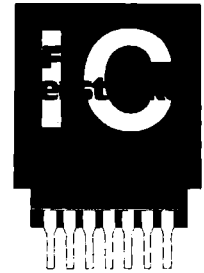
IC-Schaltungen, D. Steinbach
Hier finden Sie eine gelungene Zusammenstellung der wichtigsten Anwendungsbeispiele aus dem Bereich der integrierten Schaltungen. TTL, C MOS, Linear. Alle Schaltungen sind übersichtlich und klar dargestellt und mit einer kurzen, jedoch sehr genauen Beschreibung versehen. Tastenenprellung, Zähler, Impulsgeber, Codierer, Dekodierer, Datenübertragung, Serien-Parallel-Wandler, Digitalvoltmeter u. v. a.
Best.-Nr. 6 **19,80 DM**

Elektronik Schaltungen, Hofacker
Die ideale Schaltungssammlung zum Basteln u. Experimentieren, Schaltungen mit Operationsverstärkern, Spannungsreglern, TTL, C-MOS-Schaltkreisen. MOS Uhr mit Weckerelektronischer Würfel u. v. a.
Best.-Nr. 7 **9,80 DM**

IC Bauanleitungs Handbuch -IC -KIT, C. Lorenz
Ein Bauanleitungsbuch mit vielen hochinteressanten Bauanleitungen aus dem Bereich der LSI Schaltungstechnik. Schaltbeispiele mit Printvorlagen zum Selbstherstellen der Leiterplatten mit genauesten Beschreibungen. Hochaktuell und brandneu: Funktionsgenerator XR 2206, MOS-Uhr mit Wecker, programmierbarem Wecktongenerator, Schlummerautomatik, IC-Netzteil, Experimentieranleitung und Grundkurs über Flip Flops, u. v. a. Zu allen Schaltungen finden Sie Platinenvorlagen oder Sie können die Experimentierschaltungen auf der Experimentierplatine WH-1 g durchführen. Über 100 Seiten.
Best.-Nr. 8 **19,80 DM**

Feldeffekttransistoren, C. Lorenz
Der Feldeffekttransistor (FET) gehört heute zu den interessantesten Bauteilen überhaupt. Wie man damit experimentiert, wie man seine Funktion versteht und wie man damit brauchbare u. hochinteressante Schaltungen aufbauen kann, zeigt Ihnen dieses Buch. Grundlagen, Kennlinienfelder, Tabellen, Rechenbeispiele, Anschlußbilder und eine Vergleichsliste für Feldeffektransistoren bilden den Kern dieser umfangreichen Darstellung. Alles in allem finden Sie hier eine praxisnahe und komplette Arbietsunterlage, mit der Sie im Beruf und auch im Hobby erfolgreich arbeiten können.
Best.-Nr. 9 **9,80 DM**

Elektronik und Radio, C. Lorenz
4. Auflage. Völlig neu bearbeitet und stark erweitert. Eine Schritt für Schritt Einführung in die Radiotechnik mit vielen Bildern. Vom einfachen Diodenempfänger (Detektor) bis zu interessanten Sender- und Empfängerschaltungen (Minispione). IC-Radio, IC-Sender, Antennen, Berechnungsgrundlagen, Tabellen u. v. a. Über 150 Seiten.
Best.-Nr. 10 **19,80 DM**



NF-Verstärker, C. Lorenz
Grundlagen der integrierten NF-Verstärker, Berechnung von kompletten IC-NF-Verstärkerstufen. Anwendungsbeispiele mit den interessantesten und gebräuchlichsten Standard IC-NF-Verstärkern wie TBA 800, TBA 830, usw. Printvorlagen, Auswahltabellen, Experimentieranleitungen und Anschlußbilder machen dieses Buch zu einem unentbehrlichen Begleiter für alle, die sich m. NF-Verstärkern beschäftigen wollen.
Best.-Nr. 11 **9,80 DM**

BIS, Beispiele integrierter Schaltungen, H. Bernstein
Auf über 130 Seiten Anwendungsbeispiele mit integrierten Schaltkreisen, Zeitgeber 555, Funktionsgenerator ICL 8038, Opto Elektronik, Operationsverstärker, Festwertspeicher (ROM), u. v. a.
Best.-Nr. 12 **19,80 DM**

HEH, Hobby Elektronik Handbuch
C. Lorenz
Das Schaltungsbuch f. jeden Hobbyelektroniker, Schaltbeispiele und Bauanleitungen aus dem gesamten Hobbybereich. Lichtorgeln, Alarmanlagen, Eiswarngerät fürs Auto, PLL-Schaltungen u. v. a.
Best.-Nr. 13 **9,80 DM**

IC-Vergleichsliste, C. Lorenz
Neben integrierten Schaltungen sind auch noch Transistoren, Dioden, optoelektronische Bauteile, Sicherungen und Röhren in dieses praktische HB aufgenommen worden. Das ideale Nachschlagewerk, wenn es dann gilt das richtige Bauelement für die gewünschte Anwendung zu finden. Dieses Buch wird Ihnen in der Praxis viele Dienste erweisen können. Viele Auswahl- und Vergleichslisten mit Datenangaben. Ca. 800 Seiten.
Best.-Nr. 14 **29,80 DM**

Opto-Handbuch, C. Lorenz
Das Handbuch für die gesamte Optoelektronik. Eine Einführung und ein ideales Nachschlagwerk. Grundlagen, Definitionen aller Kenngrößen, Opto-Lexikon, Berechnungsgrundlagen, Lichtsender, Lichtempfänger, Anzeigen, Infrarot Detektoren, Optokoppler, Opto-Vergleichsliste, u. v. a.106 Seiten.
Best.-Nr. 15 **19,80 DM**

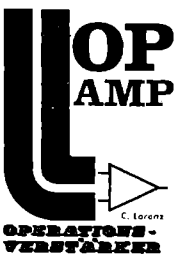
C MOS Einführung, Entwurf, Schaltbeispiele, Teil 1 H. Bernstein
Vom C MOS Gatterbaustein über Schieberegister und Zähler bis hin zum C MOS Schreib- Lesespeicher. Insgesamt werden neunzehn interessante und bekannte C MOS Schaltkreise beschrieben. Zu jedem Bauelement sind genaue Daten, Schaltbild und Anwendungsbeispiele angegeben. Im großen Applikationsteil finden Sie: C MOS-Kippstufen, Addierwerke u. Rechenschaltungen, Digital Analog Wandler, Schieberegister für analoge Spannungen, Multiplexsysteme f. analoge Signale u. v. a. Eine komplette Einführung u. gut geeignet für das Selbststudium der C MOS Technik. 140 Seiten.
Best.-Nr. 16 **19,80 DM**

Teil 2 Entwurf Schaltbeispiele

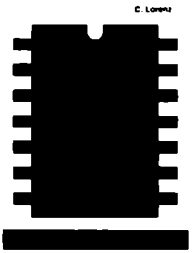
C MOS Entwurf u. Schaltbeispiele, Teil 2, H. Bernstein
Fortsetzung von Best-Nr. 16. Anwendungsbeispiele mit genauen Schaltungsbeschreibungen und Bauelementunterlagen. Daten, Anschlußbelegungen weiterer wichtiger hochintegrierter C MOS Elemente. Ein komplettes Arbeits- u. Experimentierbuch. C-MOS Uhrenschaltungen, Schieberegisterschaltungen, Parallel-Serien Umsetzung, statische u. dynamische Speicherschaltungen, Zählschaltungen, Digital Analog-Wandler, Analog Digital Wandler. Digital Voltmeter, I/O Registerschaltungen. RAM und ROM Anwendungen. Über 140 Seiten.
Best.-Nr. 17 **19,80 DM**

C MOS Entwurf u. Schaltbeispiele, Teil 3, H. Bernstein
Fortstetzung von Best.-Nr. 17. Eine sehr umfangreiche Applikationssammlung mit hochintegrierten C MOS Elementen. Speicher- und Steuerschaltungen, Multiplex- und Datenbussysteme, Liquid Cristal Anzeigen, Uhrenschaltungen, PLL-Schaltungen, Optoelektronik in Verbindung mit C MOS. Aufbau und Wirkungsweise der Prozeßrechentechnik, Arithmetische Logische Einheiten (ALU) u. andere wichtige Funktionen aus der Prozeßrechentechnik. RAMs, ROMs, und FIFO-Speicherschaltungen.
Best.-Nr. 18 **19,80 DM**

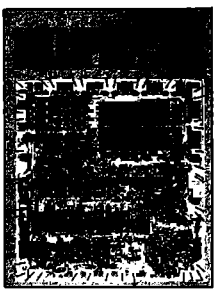
IC Experimentier Handbuch -IC, -EX, C. Lorenz
Eine sehr umfangreiche Schaltungs- und Bauanleitungssammlung mit neuen, jedoch meist beim Fachhandel erhältliche Standard ICs. Rechnerschaltungen, Mikroprozessoren, I/O-Schaltungen, Stoppuhren, druckende und anzeigende Rechner, Digitalvoltmeter, Hilfschaltungen für den Elektronik Experimentierer, A/D-Wandler, Frequenzzähler u. v. a. Viele Schaltungen können auf der IC KIT Experimentierplatine WH-1g aufgebaut werden.
Best.-Nr. 19 19,80 DM



Operationsverstärker, C. Lorenz
Dieses Buch umfaßt das gesamte Gebiet der linearen Schaltungstechnik und stellt ein in dieser Preislage bisher noch nie dagewesenes Nachschlagwerk und Einführungshandbuch dar. Bestens geeignet für das Selbststudium. Nach einer pädagogisch geschickt gemachten Einführung folgen theoretische Arbeitsunterlagen und die zugehörigen Schaltbeispiele mit Daten und Gehäuseanschlüssen. Dieses wertvolle Buch dürfte seinen Platz auch bei Ihren Arbeitsunterlagen finden, und wird dann immer von Nutzen sein, wenn es um die Lösung von nicht routinemäßigen Aufgaben geht. Über 150 Seiten.
Best.-Nr. 20 19,80 DM



Digitaltechnik Grundkurs, C. Lorenz
Ein Einführungskurs in die Digitaltechnik für Anfänger und Fortgeschrittene. Ein Fachbuch für den programmierten Selbstunterricht. Der ideale Kurzlehrgang für das Selbststudium. Der Kurs vermittelt Ihnen alle wichtigen Grundkenntnisse vom TTL-Gatter bis zum Mikroprozessor und Lösung von Schaltungsaufgaben durch Software. Viele Versuchsaufbauten u. Experimente aus diesem Kurs können auf der IC-KIT Platine WH—1g durchgeführt werden. Grundlagen, Gatter, Zähler, programmierbare Zähler, IC-Tester, Schieberegister, Speicher, Mikroprozessoren u. v. a.
Best.-Nr. 21 19,80 DM

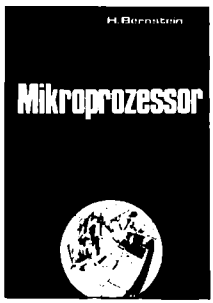


Mikroprozessoren, Eigenschaften u. Aufbau, Teil 1, H. Bernstein
Grundlagen, Eigenschaften u. Aufbau von Mikroprozessoren. Organisation von Recheneinheiten und Mikroprogr. Programmierung und Klassifizierung v. Mikroprozessoren. Ablaufdiagramm, Flußdiagramm. Ein Cip-Technik und Multi Chip-Technik, Transfer- und Sprungfunktionen. Speichertechnik: RAMs ROMs, FIFO, FILO. Programmierbare logische Arrays (PLA). Anwendungsbeispiele u. Anwendungsbereiche. Über 120 Seiten.
Best.-Nr. 22 19,80 DM

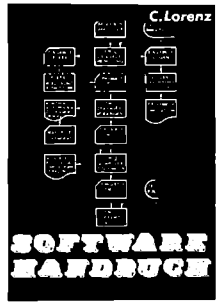
Elektronik Grundkurs, C. Lorenz
Eine leichtverständliche und pädagogisch geschickt gemachte Einführung in die Technik der elektronischen Schaltungen. Ein Kurzlehrgang und Schnellkurs zugleich. Aber auch ein recht brauchbares Nachschlagwerk für den fortgeschrittenen Elektroniker. Mit wenig Mühe können Sie sich hier die Grundkenntnisse der elektronischen Schaltungspraxis aneignen. Das Buch schafft die Voraussetzungen für ein erfolgreiches und sicheres Arbeiten mit interessanten Schaltkreisen modernster Technologien. Unentbehrlich f. das Experimentieren mit den heutigen modernen hochintegrierten Schaltkreisen.
Best.-Nr. 23 9,80 DM

Mikrocomputer Technik, Hans Peter Blomeyer-Bartenstein
Völlig neue Auflage: Herbst 1979. In diesem Buche finden Sie eine umfassende, einführende u. weiterführende Hilfe zum Einstieg in die Mikrocomputertechnik mit vielen Schalt- und Programmierbeispielen. Als praktische Betrachtungsgrundlage dient das supermoderne Microcomputerkonzept Z80A von ZILOG. Das Buch geht auf alle wichtigen Zusammenhänge ein und erklärt diese dem Leser so ausführlich, daß kaum noch Fragen offen bleiben. über 240 Seiten.
Best.-Nr. 24 29,80 DM

Hobby Computer Handbuch, C. Lorenz
Eine leicht verständliche Einführung in die Mikrocomputertechnik. Diese sehr umfangreiche Einführung in die Microcomputertechnik dürfte zu diesem Preis einmalig sein. Auf über 450 Seiten finden Sie – Grundlagen der Computer- und Microcomputer-Technik, was ist ein Microcomputer? Microcomputer KITs, Einplatinencomputer, CAT, OSI, POLY 88 u. v. a. Das ideale allumfassende Buch für den Microcomputertechniker. Für Industrieanwendung ebenso geeignet wie für den Hobby-Computer-Fan.
Best.-Nr. 25 29,80 DM



Mikroprozessor, Teil 2, H. Bernstein
Die Fortsetzung von Best.-Nr. 22. Technologie von Mikroprozessor- und Speicherbausteinen. Festwertspeicher, PROM, EPROM, FIFO, Schieberegister, MPR-, ARL- und SAR-Register. Aufbau eines Mikroprozessorsystems mit 8080, RAM- und ROM-Schnittstellen. Befehlssatz 8080. Über 120 Seiten.
Best.-Nr. 26 19,80 DM



Mikroprozessor Software Handbuch C. Lorenz
Grundlagen und Einführung in die Mikrocomputerprogrammierung und Programmiersprachen (BASIC, FORTRAN, Assembler-Sprachen). Zusammenstellung der wichtigsten Befehlslisten: 8080, Z80, M6800, 6502 etc. Ein Software Handbuch für jeden, der mit Mikroprozessoren oder Mikrocomputern zu tun hat.
Best.-Nr. 27 29,80 DM

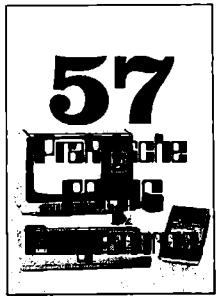
Microcomputer Lexikon u. Wörterbuch von A – Z, C. Lorenz
Einglisch/Deutsch – Der Fachausdruck wird übersetzt, ausführlich erklärt und erläutert. Deutsch/Englisch – Übersetzung des Fachausdrucks. Ein Hilfs- und Arbeitsbuch für jeden, der sich heute mit der modernsten Elektronik beschäftigt. Viele engl. Ausdrücke werden heute in der Elektronik, Computer- und Mikroprozessortechnik verwendet und oft fehlt uns eine genaueste und präzise Erläuterung. Ein Lexikon und Wörterbuch in einem einzigen Buch vereint.
Best.-Nr. 28 29,80 DM

ohne Abbildung

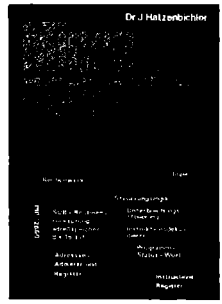
Erscheint ca. Ende 1981

Mikrocomputer Datenbuch, Lorenz
Eine übersichtliche u. sehr informative Zusammenstellung der wichtigsten Mikroprozessorbausteine auf dem Markt. 8080A, 8085, 8048, Z80, Z8, 6500, 6800, 2650, 1802, F8, 3870, 6809, PACE u. v. a. Daten, Anschlußbilder, wichtige technische und elektrische Daten, Architektur, grundlegende Eigenschaften. Zu jedem Mikroprozessor werden dann auch noch die peripheren Bausteine sowie RAM und ROM Elemente behandelt. Das ideale Handbuch für jeden modernen Elektroniker. Ideal f. den Computer-Service. (Englisch)
Best.-Nr. 29 49,80 DM

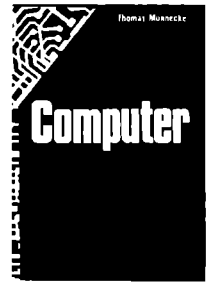
Aktiv Training Microcomputer 8080 8085, S. Lehrer
Dieses Werk mit über 360 Seiten beschäftigt sich ausschließlich mit den Microcomputerbausteinen und Peripherieelementen der 8080A und 8085 Microprozessoren. Grundlagen Einführung 8080 Microcomputer, Programmieren in Assebmler, PL/M Cimpiler, Speicherbausteine u. v. a. Ideal für jeden, der ein System mit 8080, 8085 CPU besetzt. Auch der Z-80 Systembesitzer kann von diesem Buch viel profitieren. Am Schluß des Buches finden Sie noch ca. 150 Programmlistings in Maschinensprache (Nützliche Utilities und Spielprogramme).
Best.-Nr. 30 49,80 DM



57 Programme in BASIC, Lorenz
Ein Buch mit techn.-wissenschaftlichen Programmen u. einer großen Anzahl von Spielprogrammen in BASIC (Games). Ein Buch für jeden, der sich mit dem faszinierenden Hobby der Mikrocomputertechnik befassen will. Alle Listings sind in BASIC und können auf den meisten Personal Computer Systemen gefahren werden.
Best.-Nr. 31 **39,00 DM**



Mikrocomputer Programmierbeispiele für 2650, Dr. J. Hatzenbichler
Eine Einführung in die Programmierung von Mikrocomputern anhand des Prozessors 2650 von Signetics. Viele Programmierbeispiele in Maschinensprache, die Sie auf einem preiswerten Mikroprozessorsystem MIKIT 2650-P2 ausführen können. Zeitschleifenprogr., Blinkschaltung, Lauflicht, Stufenzähler, Stopuhr, Reaktionszeittester, u. a. Zu diesem Buch ist auch ein komplett aufgebautes und getestetes Mikrocomputersystem erhältlich, auf dem Sie alle beschriebenen Programme selbst ausführen können. Über 120 Seiten. (Nur solange Vorrat reicht, wird nicht mehr nachgedruckt.)
Best.-Nr. 33 **19,80 DM**



Der freundliche Computer – Was können Sie mit einem Personal Computer anfangen? T. Munnecke
Das Buch soll Ihnen auf die im Titel gestellte Frage eine ausführliche Antwort geben. Es eignet sich für alle, die bisher viel über Mikros gehört haben und gerne ausführlicher Bescheid wissen möchten. Viele interessante Fakten – Welche Computersprache ? Welche Anwendungen ? Welches Gerät soll ich mir kaufen ? 153 Seiten.
Best.-Nr. 35 **29,80 DM**

ohne Abbildung

Microcomputer und Roboter
Ein Buch für denjenigen, der sich externe Schaltungen für Microcomputer bauen möchte, die roboterartige Funktionen ausführen können. Spracherkennung, Analog-/Digital-Wandler, Digital-/Analog-Wandler, Ultraschallsensoren, Lichtschranken, Tonerzeugung u. v. a. Erscheint Anfang 1982.
Best.-Nr. 36 **29,80 DM**

ATARI BASIC Handbuch
Das komplette Einführungs- und Programmierhandbuch für die neuen ATARI-Computer ATARI 400/800. Anhand von vielen Beispielen wird die Leistungsfähigkeit des ATARI-Computer-Systems gezeigt. Auch für den Anfänger ideal als Einstieg gedacht. Das Buch kann auch als Kurs (Lehrgang) für die allg. BASIC-Sprache verwendet werden.
Best.-Nr. 32 **29,80 DM**

TINY BASIC Handbuch, Hermann
Das erste deutschsprachige Handbuch über Tom Pittman's TINY BASIC. Eine Einführung in die TINY BASIC-Programmiersprache. Wie kann ich meinen Computer (KIM-1) erweitern und BASIC programmieren. Systemvorschläge. Viele Programmierbeispiele, Tricks und Kniffe.
Best.-Nr. 34 **19,80 DM**

ohne Abbildung

Oszillographen Handbuch
Ein Buch für jeden, der seinen Oszillographen optimal nutzen will. Der Anfänger, der noch nie einen Oszillographen benutzt hat, wird auf einfache Weise mit der Technik und Handhabung vertraut gemacht. Auch die Anwendung in Zusammenhang mit den modernen Mikrocomputersystemen wird beschrieben. Oszillograph als alphanumeres Darstellgerät u. v. mehr. Erscheint ca. Anfang 1982.
Best.-Nr. 103 **19,80 DM**

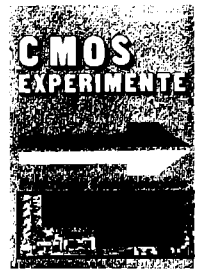
ohne Abbildung

1000 Elektronik Schaltungen
Ein ideales Handbuch für jeden, der öfter eine Schaltung zur Lösung eines bestimmten Problems sucht. Tausend Schaltungen aus fast allen Bereichen der Elektronik. Industrielle Steuerschaltungen, Microcomputer, Peripherie, Hobby-Elektronik-Schaltungen u. v. a. mehr. Ein Buch, daß bei keinem Elektroniker fehlen sollte. Erscheint ca. Mitte 1982.
Best.-Nr. 104 **49,00 DM**

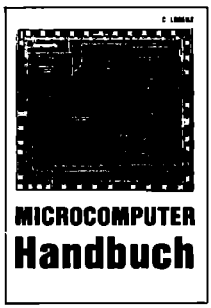
ohne Abbildung

Praktische Antennentechnik
Ein Buch für jeden Funkamateur oder Hobbyfunker. Grundlagen, Einführung, praktische Besipiele, Berechnungsgrundlagen u. v. a. Erscheint ca. Anfang 1982.
Best.-Nr. 107 **19,80 DM**

TTL-Experimentierbuch
Eine kleine Einführung in die Digitaltechnik. Grundlagen der Digitaltechnik kurz erklärt. Bauanleitung für einen praktischen Logik-Tester. Viele Experimente u. Anwendungsschaltungen mit dem TTL Gatterbaustein 7400. Das ideale Einführungsbuch in die Digitaltechnik.
Best.-Nr. 105 1 **5,00 DM**



CMOS-Experimentierbuch
Eine kleine Einführung in die CMOS Schaltungstechnik. CMOS Grundlagen, Behandlungshinweise für CMOS Bausteine, Zusammenst. der wichtigsten CMOS Bausteine und deren Anschlußbilder. Viele praktische CMOS Schaltbeispiele. Ideal für jeden Elektroniker.
Best.-Nr. 106 1 **5,00 DM**



Handbuch für SC/MP
Ein echtes Handbuch für SC/MP (INS8060) Microcomputerbesitzer und solche die es werden wollen. Komplette Einführung mit vielen Schaltbeispielen, Schaltbildern und Programmlistings in Maschinensprache und TINY BASIC. Komplettes ELBUG-Listing, CPU-Karte, RAM-Karte, ROM-Karte, ROM-Programmierer, Cassetten-Interface, Fernsehinterface, Netzteil, Hex Ein-/Ausgabe, SC/MP Einkartenmicrocomputer u. v. a. 330 Seiten.
Best.-Nr. 108 **29,80 DM**

6502 Microcomputer Programmierung, P. Heuer
Eine deutschsprachige Einführung in die Maschinensprachenprogrammierung anhand des 6502 Microcomputers. Ein echtes Anleitungsbuch zum Einstieg in die Microcomputertechnik mit Hilfe des KIM-1. Viele Programmierbeispiele, die von einem Pädagogen speziell für Anfänger entwickelt wurden. Auch PET, AIM, SYM und ATARI-Besitzer brauchen dieses Buch.
Best.-Nr. 109 **29,80 DM**

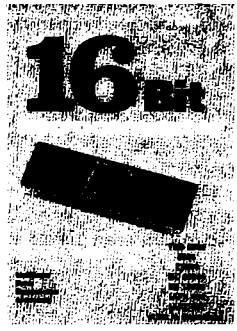
Programmierhandbuch für PET
C. Lorenz
Ein für den PET produziertes Buch. Es beginnt da, wo Ihr mitgeliefertes Handbuch aufhört. Einführung Maschinenprogrammierung, Assembler, Ein-/Ausgabeprogrammierung, Programmiertricks, Analog / Digital-Wandler, Graphik, Spracherkennung u. v. a. mehr. Viele Listings, die Sie selbst eintippen können. Viele Informationen eignen sich auch für VC—20-Besitzer. 324 Seiten.
Best.-Nr. 110 **29,80 DM**

Programmieren mit TRS-80, Stübs
Das erste in einem deutschen Verlag produzierte Buch über den erfolgreichen Personal Computer von TANDY. Ein Buch für jeden, der einen TRS-80 bereits besitzt oder vor der Entscheidung steht, welchen Computer er sich anschaffen soll. Einführung, Programmiertricks, Erweiterungen, Maschinenprogrammierung und viele Programme (Listing mit Beschreibung). 202 Seiten.
Best.-Nr. 111 **29,80 DM**

BASIC Programmierhandbuch
Einführung und Nachschlagewerk. Speziell für die BASIC-Versionen der modernen Microcomputersysteme. Jeder Befehl wird ausführlich beschrieben und ein Beispielprogramm gezeigt. Sehr übersichtlich und praktisch. Am Schluß finden Sie ein komplettes BASIC-Programm, das Ihnen über einen Computer BASIC lehrt.
Best.-Nr. 113 **19,80 DM**

Der Microcomputer im Kleinbetrieb
Das Buch für jeden Geschäftsmann. Auf über 170 Seiten erfahren Sie, was Sie als Gewerbetreibender oder freiberuflich Tätige über Microcomputer und die Anwendung wissen sollten. Geschichtlicher Hintergrund Geräteauswahl, Beispiele aus der Praxis, Programmbeispiele wie z. B. Textverarbeitung, Reisebüro, Ladenkasse, Adressverwaltung, u. v. a. Betriebswirtschaftliche Auswertung, Finanzbuchhaltung, Erfolgsanalyse mit dem Microcomputer, Liquiditätsrechnung, kurzfristige Erfolgsrechnung, Microcomputer für Freiberufler, Grundlagen der Finanzbuchhaltung für Microcomputeranwender. Dieses Buch kann Ihnen als Geschäftsmann für die Zukunft tausende einsparen.
Best.-Nr. 114 **39,80 DM**

PASCAL Handbuch, E. Flögel
Von BASIC zu PASCAL. Ein Einführungs- Lehr und Arbeitsbuch für jeden der sich mit PASCAL beschäftigen will oder muß. Viele Programmbeispiele, viele Tricks wie PEEK und POKE, Einbinden von Maschinenprogrammen u. v. a.
Best.-Nr. 112 **29,80 DM**



16 Bit Microcomputer, J. Koller
Einführung, Daten, Eigenschaften, Anwendungen. Dieses Werk ist eine echte Sensation !
Alle 16 Bit Prozessoren werden beschrieben und erläutert. Applikationsbeispiele, Programmierhinweise. TMS 9900, 8086, Z8000, MC 68000, NS 16000, IAPX 486, IAPX 432. Über 370 Seiten.
Best.-Nr. 116 **29,80 DM**

ohne Abbildung

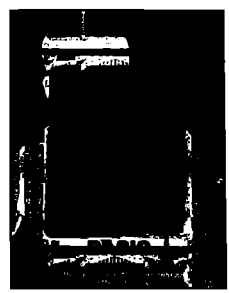
FORTRAN für Heimcomputer
Einführung in die FORTRAN-Programmiersprache mit vielen Beispielen. Grundsätzliches über die verschiedenen Microcomputersysteme, die bereits mit FORTRAN-Compiler lieferbar sind. Allgemeine Übersicht, Tips und Hinweise.
Erscheint ca. Ende 1982 .
Best.-Nr. 117 **19,80 DM**

Programmieren in Maschinensprache mit 6502, E. Flögel, W. Hofacker
Das deutschsprachige Werk über 6502 Maschinenprogrammierung. Einführung, Grundlagen, Eigenschaften, Adressierungsarten, Befehlsarten. Wie entwickelt man ein 6502 Maschinenprogramm? Handassemblierung, viele Programmbeispiele mit genauen Angaben direkt zum Eingeben in den Apple II mit Adressangabe (keine blutleeren Beispiele ohne Adresse), Verwendung von Assemblern. AIM-Assembler, Disassembler, Relocator, 6522 VIA, 6520, Interrupt, Fehlersuche in Maschinenprogrammen, Maschinensprache, Programmiertricks. Spezielle Abschnitte für Maschinensprachenprogrammierung über PET, CBM 3000, CBM 4000 u. VC-20, ATARI 400/800, Apple II, AIM sowie Ohio Scientific Challenger. Dieses Buch sollte jeder 6502 Systemanwender besitzen. Ca. 240 Seiten.
Best.-Nr. 118 49,00 DM

Anwenderprogramme für TRS-80 von Martin Stübs
Ein Buch, voll mit interessanten Anwenderprogrammen für TRS-80 Level II 16K und Video Genie (teilweise Diskette u./od. Cassette). Hauptsächlich Programme für den Manager, Geschäftsmann, Klein- und Mittelbetrieb. Auch einige interessante Spiele sind enthalten. Terminkalender, Reservierungsprogramm für Omnibusunternehmen und Hotels, Textverarbeitung, usw.
Best.-Nr. 120 29,80 DM

BASIC für Fortgeschrittene
Endlich ein BASIC-Buch für den fortgeschrittenen Programmierer. Alle wichtigen Befehle aus der Stringmanipulation, Disk-Befehle, WAIT, INSTR, WHILE WHEND usw. werden an Beispielen besprochen. Alle Befehle sind übersichtlich wie in einem Nachschlagwerk mit großen Überschriften angeordnet. Dann folgt ein umfangreicher BASIC Kurs für Fortgeschrittene mit vielen Beispielen (Inventur, Rechnungen schreiben, Adressenverwaltung usw.). Am Schluß finden Sie dann noch einen Vergleich der wichtigsten Sortiermethoden sowie ein Programm zur Vorhersage von Ereignissen.
Best.-Nr. 122 39,00 DM

Programmieren in Maschinensprache (Z80), C. Lorenz
Eine sehr ausführliche Einführung in die Z80 Maschinensprache mit vielen Beispielen. Die Beispiele können mit Hilfe des TRS-80 Level II sowie dem T-BUG von TANDY und den T-BUG-Erweiterungen (IN LOCO, T-STEP, T-LEGS) ausgeführt werden. Ein unentbehrliches Buch für jeden, dem die BASIC-Programmiersprache von der Geschwindigkeit her zur Lösung seiner Aufgaben nicht mehr ausreicht.
Best.-Nr. 119 49,00 DM

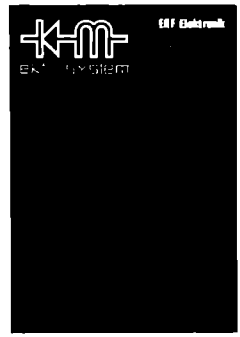


Microsoft BASIC-Handbuch
Die deutsche Übersetzung des erfolgreichen Microsoft BASIC-Handbooks. Leicht verständliche Einführung mit vielen interessanten Programmbeispielen. Das kompetente Werk von Microsoft selbst. Ideal als Zusatzliteratur zu jedem BASIC-Buch.
Best.-Nr. 121 29,80 DM

IEC Bus-Handbuch, M. P. Gottlob
Ein Handbuch und Nachschlagewerk für alle Besitzer von Computern mit IEC (IEEE 488 Bus). Dazu gehört auch der PET sowie alle CBM-Computer. Grundlagen, das BUS-System, Meßdatenübertragung, Adressierung eines Instruments, kl. IEC-BUS-Lexikon u.v.a.
Best.-Nr. 123 **19,80 DM**

Programmieren in Maschinensprache mit CBM
An Hand eines praktischen Beispiels (Sortierroutine) wird der Unterschied zwischen BASIC und Maschinenprogrammen gezeigt. Das Maschinenprogramm kann mit dem leistungsfähigen MONJANA/1 Monitor in ROM erstellt werden. Am Schluß finden Sie weitere wichtige Informationen wie Dez/Hex-Umrechnungstabelle, Befehlslisten, ASCII-Tabelle sowie eine ROM-Vergleichsliste zwischen 8k PET und den neuen CBM-Maschinen.
Best.-Nr. 124 **19,80 DM**

MONJANA Monitor im ROM
Best.-Nr. 1241 **79,00 DM**



Einführung in die Microcomputer Programmierung mit 6800
Eine sehr gute Einführung in die Microcomputertechnik mit Hilfe des Mikroprozessors 6800. Ausführliche Erklärungen mit vielen Beispielen und Anleitungen. Theoretische Grundlagen. CPU-Architektur, Befehlssatz, Systemaufbau, Hilfsmittel der Programmierung, Trainingsprogramme, Systemkomponenten, FIRMWARE. Ein komplettes Monitorprogramm (Betriebssystem) ist als Listing enthalten. Über 250 Seiten.
Best.-Nr. 127 **49,00 DM**

Programmierbeispiele für CBM
Ein Buch mit vielen BASIC-Programmen für CBM und PET. Spiele, Geschäftsbereich, Erziehung und Wissenschaft, Utilities, Hilfen für Maschinensprachenprogrammierung, trickreiche Programme. Viele Programme für wenig Geld.
Best.-Nr. 130 **19,80 DM**

ohne Abbildung

ELCOMP, Fachzeitschrift für Microcomputertechnik
Die kompetente Fachzeitschrift für das moderne Gebiet der Microcomputertechnik. Erscheint 10 x pro Jahr. Jahrespreis *** 59,00 DM** incl. MwSt., Porto und Verpackung. Wer die neuesten Informationen aus diesem Gebiet für sich nützen möchte, muß ELCOMP lesen. Software, Technische Tips, Programmiertricks, Bauanleitungen, Systembeschreibungen, u. v. a. Jeden Monat brandneu. ISSN-Nr. 0171-0958
Best.-Nr. 125 **je Heft * 4,50 DM**

ELCOMP-Doppelheft
Von den 10 Heften erscheint im Juli/August und November/Dezember ein Doppelheft.
Best.-Nr. 126 *** 9,00 DM**

* ab 1.1.1982 gelten die neuen Preise:
Abbonement 69,00 DM, Einzelheft 5,00 DM
Doppelheft 10,00 DM

ohne Abbildung

Programmieren mit dem CBM
Ein Hand- und Programmierbuch für alle CBM-Besitzer der 3000, 4000 sowie der 8000er Serie. Viele Tricks und Programmierbeispiele, Anleitungen. Erscheint Ende 1981.
Best.-Nr. 128 **29,80 DM**

ohne Abbildung

ELCOMP Leser Programmierhandb.
Hier fassen wir die besten Programme unserer ELCOMP-Leser zusammen. Programme für PET, CBM, TRS-80, AIM, Superboard, C4P, Exidy, Sharp, MZ80K, Apple II, Nascom I und II und TI 99/4 werden als Listing mit kurzer Beschreibung allen Lesern zugänglich gemacht. Erscheint Anfang 1982.
Best.-Nr. 129 **69,00 DM**

ohne Abbildung

Cobol für Anfänger
Eine Einführung in die Cobol-Programmierung für den Microcomputer-Besitzer. Erscheint ca. Mitte 1982.
Best.-Nr. 131 **19,80 DM**

ohne Abbildung

CP/M Handbuch
Grundlagen, Einführung, Hilfs- und Handbuch für jeden der mit dem "Software-Bus" arbeiten möchte. Ideal auch für Anfänger. Praktisches Handbuch für den Profi. Erscheint ca. Ende 1981.
Best.-Nr. 132 **19,80 DM**

ohne Abbildung

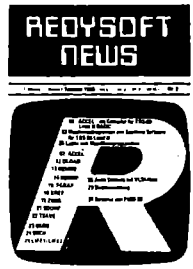
Welches Betriebssyst. brauche ich?
Ein Leitfaden und Handbuch für den Anwender der gehobenen Microcomputerklasse. UNIX, CP/M, OASIS, NORTHSTAR DOS, USCD' Apple DOS, ATARI DOS, CP/A, usw. Erscheint Mitte 1982.
Best.-Nr. 133 **19,80 DM**

ohne Abbildung

Microcomputer im Unterrichtsfeld
M. Penzkofer
Wie werden Microcomputer im Schulbetrieb eingesetzt? Grundlagen Wissenswertes, Tips, Erfahrungen und viele Beispiele. Für alle die die moderne Microelektronik für Schule Ausbildung und Erziehung nutzen wollen. Erscheint ca. Anfang 1982.
Best.-Nr. 136 **29,80 DM**

ohne Abbildung

FORTH Handbuch und Einführung
von E. Flögel, W. Hofacker
FORTH ist nicht nur eine sehr leistungsfähige Programmiersprache — es ist schon fast eine "Religion". FORTH eignet sich bestens für industrielle Steuerungen, Grafik etc. Grundlagen und viele Programmbeispiele (Apple II, Ohio, ATARI usw.). Erscheint Mitte 1982.
Best.-Nr. 137 **39,00 DM**



Redysoft News
Das Magazin für den Microcomputereinsatz im kommerziellen Bereich. Redysoft-News erscheint in zwangsloser Folge. Bisher sind drei Hefte erschienen. Aus dem Inhalt:
Heft 1: Microcomputer für freiberuflich Tätige. Die Anwendung von Microcomputern im Geschäftsbereich. Microcomputer-Markt. Ein Microcomputer zum Arbeiten. Die Microschule. MICROS werden voll geschäftsfähig. Challenger Superboard, der ideale Einstieg.
Heft 2: ACCEL - ein Compiler f. TRS-80 Level II BASIC. Maschinenprogramme von Southern Software für TRS-80 Level II
Laden von Maschinenprogrammen ACCEL, DLOAD, RENUM, HDUMP, FGRAF, XREF, ZBUG, SDUMP, TSAVE, USRN, SRCH, LIFE1 + LIFE2. Auch Siemens auf VLSI-Kurs. Dateiverwaltung. Struktur von FIBU-80.
Heft 3: APPLE Utility Disk. Bilanzbuchhaltung u. Finanzbuchhaltung f. den Microcomputer TRS-80 Model II. ACCEL u. ACCEL2 ? - BASIC-Compiler für den TRS-80 Model I. Eine Spielcassette für den TRS-80. Adressenverwaltung. Disassembler. TAPECHECK. SAVE. Terminkalender. Lagerverwaltung. ADEKART. Computerspiele für höchste Ansprüche.
Best.-Nr. 135 **je Heft** **2,00 DM**

ohne Abbildung

ADA - Handbuch, **E. Flögel**
Vorankündigung für Mitte 1982. Mit einer praktischen und leicht verständlichen Einführung soll hier dem Leser diese neue Programmiersprache nähergebracht werden. APA wurde, wie Cobol vom US-Verteidigungsministerium entwickelt und soll die steigenden Kosten bei der Softwareentwicklung eindämmen. ADA ähnelt in gewisser Weise d. Programmiersprache PASCAL und dürfte bald zum neuen Standard werden. Zur Zeit (Mitte 1981) gibt es jedoch noch keinen Microcomputer auf denen ADA implementiert ist.
Best.-Nr. 138 **29,80 DM**

ohne Abbildung

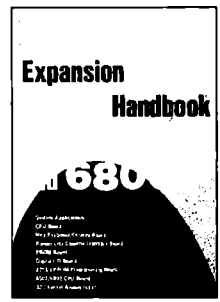
BASIC für blutige Laien
von W. Hofacker
Endlich ein Buch für den Anfänger und Laien. Ziel des Buches ist es, dem "blutigen Laien" die Grundlage der Programmiersprache BASIC zu vermitteln. Lernen wird nun zum echten Vergnügen und Freizeitspaß. Auch der Preis macht Spaß. Erscheint ca. Ende 1981.
Best.-Nr. 139 **nur** **19,80 DM**

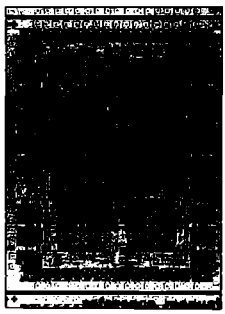
ELCOMP-Bücher Englisch

Care and Feeding of the Commodore PET
Das ideale Buch für den Hardware-Bastler. Viele Tricks, Schaltbilder, Hinweise und Erläuterungen für den, der gerne selbst Erweiterungen bauen möchte. Memory Map für 8k PET und CBM, Bauanleitung für eine serielle Schnittstelle u. v. a.
Best.-Nr. 150 19,80 DM

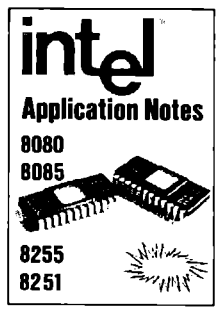
Complex Sound Generation with SN 76477
Ein Applikationsheft für einen der interessantesten integrierten Bausteine unserer Zeit. Ein LSI-Baustein zur Tonerzeugung. Je nach äußerer Beschaltung können Sie mit diesem Baustein die verrücktesten Töne erzeugen. Dampfeisenbahngeräusch mit Dampfpfeife, Vogelgezwitscher, Hundegebell, elektronische Orgel, Schuß mit Explosion u. v. a. mehr.
Best.-Nr. 154 19,80 DM

Expansion Handbook for 6502 and 6800
Das ideale Handbuch für alle KIM, SYM, AIM, PET und Challenger Computer-Freunde. Das Buch beschäftigt sich ausschließlich mit dem S-44-Bus. Dies ist exakt der Bus von SYM, AIM und KIM. Sehr viele Schaltbilder: CPU-Platine, Hex-Tastatur Eingabe, Knsas City Interface, RAM u. ROM-Karte, Analog-Eingabe Board u. v. a. Das Buch ist für jeden 6502 Systembesitzer unentbehrlich. Ca. 150 Seiten.
Best.-Nr. 152 19,80 DM

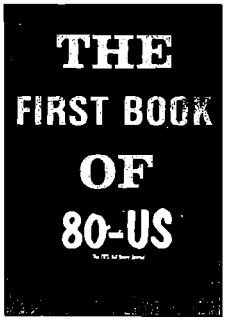




Microsoft 8k BASIC Reference Manual
Eine sehr gute BASIC-Einführung. Auch als Handbuch zum Nachschlagen bestens geeignet. Ideal für jeden PET, CBM, TRS-80, KIM-BASIC, SYM-BASIC, AIM- und APPLE-Besitzer. 73 Seiten DIN A4 mit vielen Beispielen.
Best.-Nr. 151 19,80 DM



Intel Application Notes (8080, 8085, 8255, 8251)
Dieses Buch braucht jeder, der mit 8080, 8085 oder Z-80 Mikroprozessoren arbeitet.
Wir haben die interessantesten Applikationsberichte in diesem Buch zusammgefasst. Aus dem Inhalt: Designing with Intel's Static RAM's 2102, Memory Design with the Intel 2107B. 8255 Programmable Peripheral Interface Applications, Using the 8202 Dynamic RAM Controller u. v. a.
Best.-Nr. 153 29,80 DM

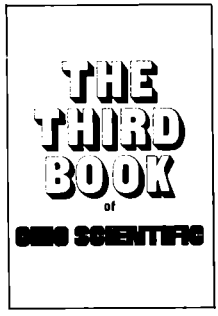


The First Book of 80-US
Für den TRS-80 Freund eine echte Preissensation. Die ersten fünf Hefte aus 80-US Journals in einem Sammelband zusammengefaßt. Voll mit vielen sehr interessanten Hard- und Softwareideen, Tricks. Viele komplette Programmbeispiele (Listings) in BASIC u. Z-80 Maschinensprache. Über 250 Seiten DIN A4. Farbiger Umschlag. Dieses Buch sollte jeder TRS-80 Besitzer oder der es werden will im Schrank haben.
Best.-Nr. 155 **29,80 DM**

Small Business Programs, S. Roberts
Ein Buch für denjenigen, der die modernen Microcomputer (speziell TRS-80, Apple, PET, North Star, Challenger) zur Rationalisierung in seinem Klein- oder Mittelbetrieb einsetzen möchte. Viele nützliche Tips, Hinweise und Programmierbeispiele. Dieses Buch sollte jeder Geschäftsmann u. Microcomputerfreund besitzen.
Best.-Nr. 156 **29,80 DM**

The first Book of Ohio Scientific
Das erste weltweit produzierte Buch für die erfolgreiche Ohio Scientific Challenger Computerserie. Grundlagen, viele Programmiertricks Hardwaretips, Umbauanleitungen, Programmierbeispiele u. v. a. Glanzumschlag, 186 Seiten.
Best.-Nr. 157 **19,80 DM**

The second Book of Ohio Scientific
Eingehende Beschreibungen über praktische und geschäftsorientierte Software. Speicher Test Programm, Tricks und Tips für Disketten-Anwender. Mini-Floppy-Expansion u. v. a. 159 Seiten.
Best.-Nr. 158 **19,80 DM**



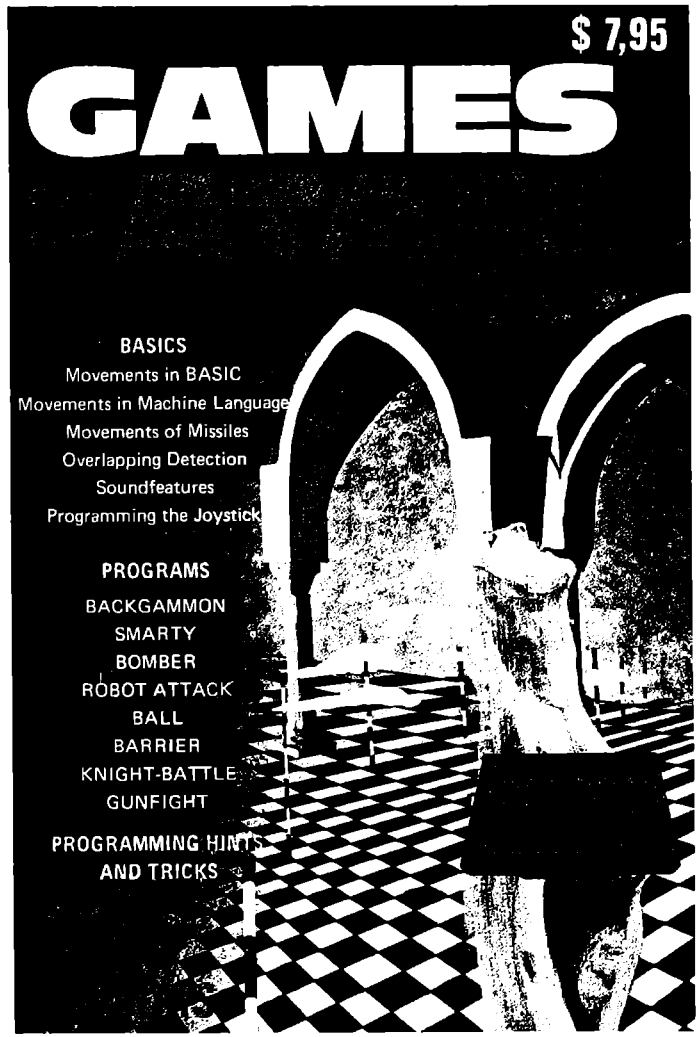
The third book of Ohio
Wie erweitere ich mein Challenger System ? Universelle I/O-Karte, EPROM-Burner für 2716, EPROM, RAM-Karte, 6522 VIA-Karte. Wo notwendig mit kompletter Software. Dieses Buch braucht jeder Ohio-Benutzer. Komplette Schaltbilder und Aufbauhinweise.
Best.-Nr. 159 **19,80 DM**

The fourth Book of Ohio Scientific
Ein Buch voll mit Programmen für das Superboard, C4P, C4PMF und C28P. Die Softwarequelle für jeden Challenger-Fan. Alle Programme sind getestet und auch auf Cassette verfügbar. 170 Seiten Listings und Beschreibungen.
Best.-Nr. 160 **29,80 DM**
Best.-Nr. 8324 Cassette **29,80 DM**

ohne Abbildung

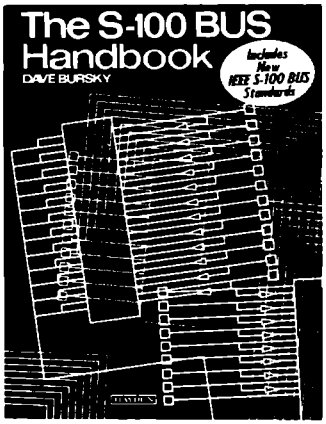
The fifth Book of Ohio Scientific
Fortsetzung der beliebten Reihe über die Ohio Scientific Computer. Hier liegt der Schwerpunkt wieder bei der Software. Viel interessante Software. Eine komplette Adressenverwaltung mit Label und Listenausgabe. Mehrere Fakturierprogramme, viele interessante Spiele wie Autorennen, Tankwar, Tonerzeugung, Joystickprogrammierung usw. Erscheint ca. Anfang 1982.
Best.-Nr. 161 **19,80 DM**

ATARI-BASIC - Learning by Using von Tom Rowley
Ein sehr interessantes Lehr- und Programmierbuch für jeden ATARI-Besitzer. Kurze Programme und Lernbeispiele machen das Programmieren zum Kinderspiel. Viele Programme in diesem Buch verwenden weiterführende raffinierte Programmiertechniken andere wiederum sind einfach und ideal für den Erstkontakt mit dem ATARI-Mikrocomputer. Tonerzeugung, Grafik, Joystickprogrammierung, Player Missle Graphic, Video-Kunst u. v. a. Ca. 75 Seiten.
Best.-Nr. 164 **19,80 DM**

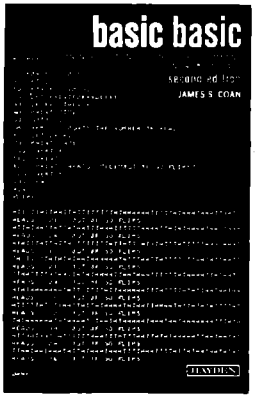


Games for the ATARI 400/800, Sam Roberts
Wie programmiere ich meine eigenen Spiele auf meinem ATARI Privatcomputer? Dieses Buch zeigt Ihnen auf einfache Weise den Weg in die wunderbare Welt der Spiele — Programmierung.
Sie nutzen voll die phantastischen Grafik und Tonausgabe-Möglichkeiten des ATARI-Computers. Viele kleine und sehr lehrreiche Programmbeispiele machen dieses Buch zu einem echten Erlebnis. Eine Vielzahl von umfangreichen BASIC-Programmen wie Robot Attack, Bomber, SMARTY, BACKGAMMON usw. können Sie aus dem Buch direkt in Ihren Computer eintippen und abspeichern. Ein komplettes AKTION Spiel in Maschinensprache ist als kommentierter Quelltext und Hexdump im Buch vorhanden. Dieses Spiel mit dem Namen Gunfight wird Sie begeistern. Auf ca. 128 Seiten Viel Information. **Best.-Nr. 162** **nur 19,80 DM**

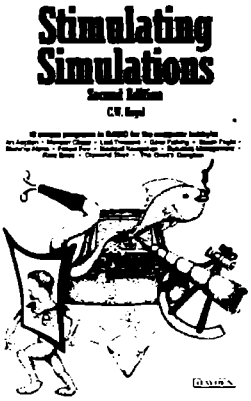
Hayden-Bücher



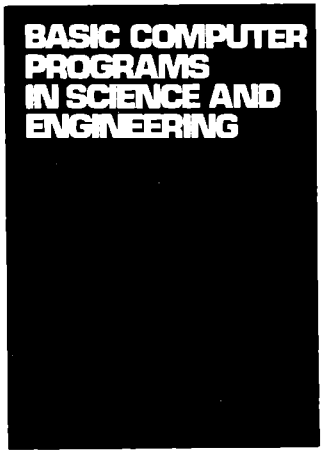
The S-100 Bus Handbook
(enthält den Neuen S-100 Standard). Alles was Sie über den S-100 Bus wissen sollten. Einführung, Schaltungen. Schaltbilder der oft verwendeten S-100 Platinen. Service u. Fehlersuche in S-100-Systemen, Interfacebeispiele u. v. a. 256 Seiten Großfomat.
Best.-Nr.: 254 **49,00 DM**



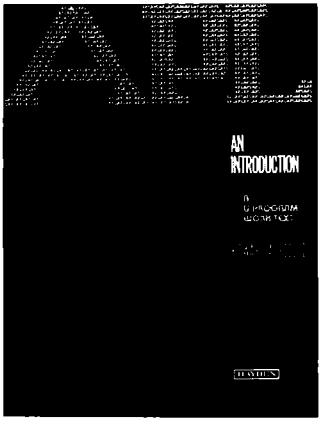
BASIC BASIC
Eine Einf. in die BASIC-Programmierung v. Microcomp. Eines der besten BASIC-Bücher weltweit. Viele Beispiele aus dem professionellen u. wissenschaftl. Bereich.
Best.-Nr.: 255 **39,00 DM**



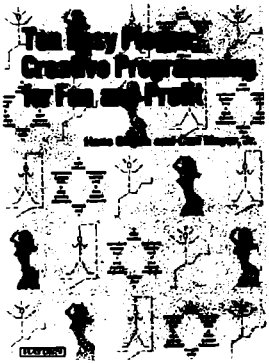
Simulating Stimulations, v. C. W. Engel 12 einzigartige Programme in BASIC f. d. Comp.-Hobbyisten. (Kunstauktion, Monster Chase, Lost Treasure, Gone Fishing, Space Flight, Starship, Forest Fire, Navigation, Business Management, usw.
Best.-Nr.: 256 **19,80 DM**



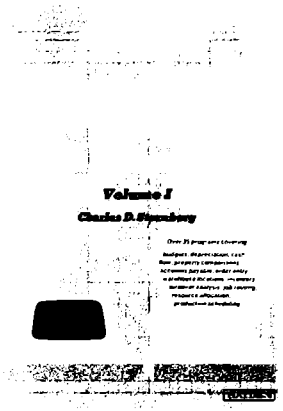
BASIC Computer Programs in Science and Engineering. (250 Seiten Großformat) Programme aus den verschiedensten Bereichen m. genauen Beschreibungen. Komplexe Zahlen, Engineering, Mathematik, Matrizen, Analyse von Daten, Elektrotechnik, Electronic, Filterberechnung usw.
Best.-Nr.: 257 **39,00 DM**



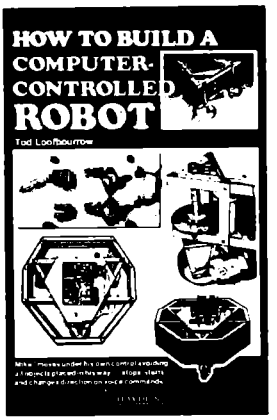
APL - An Inroduction
Ein Programmier- und Arbeitsbuch. Ein Einführungsb. zum Selbststudium von APL. Viele Beisp., Übungsbeisp. m. Lösungen. 245 Seiten Großformat.
Best.-Nr. 258 **39,00 DM**



Ten Easy Pieces:
Creative Programming for Firm and Profit
10 Beispielhaft beschriebene BASIC-Programme. Schritt f. Schritt wird jeder Befehl u. Zusammenhang erläutert. Schwerpunkt Simulationsprogramme. Sehr gut geeignet f. Ausbildung, Selbststudium und als Nachschlagewerk.
Best.-Nr. 259 **29,80 DM**



BASIC Computer Programs für Business, Vol. I Charles D. Sternberg
Über 35 Programme aus dem Gebiet : Budget, Abschreibung, Cash-Flow, Grundstückswertvergleich, Accounts Payable, Auftragseingang, Inventur-Umschlagsanalyse, Produkt-Terminplanung u. v. a. Alle Programme sind ausführlich beschrieben. Ein sehr wertvolles Buch. 250 Seiten, Großformat
Best.-Nr. 260 **39,–**



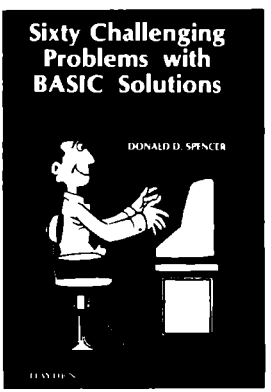
How to build a Computer controlled Robot, v. Tod Loofbourrow
Komplette Bauanleitung für einen mit KIM-1 gesteuerten Roboter. Viele Schaltungen A/D- D/A-Wandler mit Software. (Ultraschall-Detektion, Spracherkennung u.v.a)
Best.-Nr.: 253 **35,00 DM**

Home Computers Can Make Your Rich
Ein Experte zeigt Ihnen wie man mit Personal-Computern Geld verdienen kann. Was ist zu beachten? Viele Ideen.
Best.-Nr. 262 **19,80 DM**

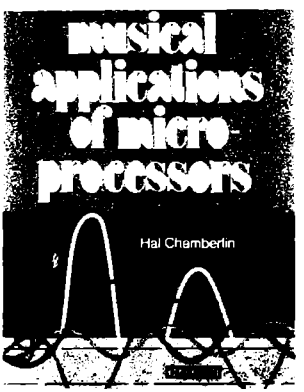
ohne Abbildung

BASIC Computer Programs For Business
Best.-Nr. 261 **39,00 DM**

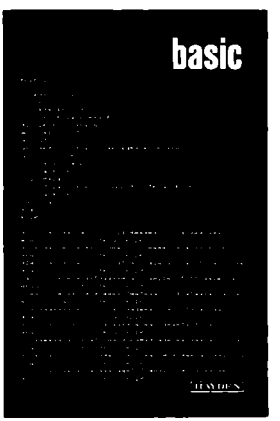
The complette 1802 Cookbook
Best.-Nr. 264 **19,80 DM**



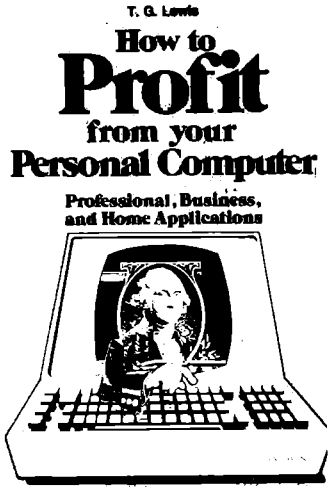
Sixty Challenging Problems with BASIC Solutions. 60 Spiele, Puzzles, mathematische und wissenschaftliche Problembeschreibungen, die der Programmierer in BASIC lösen soll. Am Ende des Buches sind zur Kontrolle alle Lösungen (60 BASIC Programme) aufgeführt.
Best.-Nr.: 263 **19,80 DM**



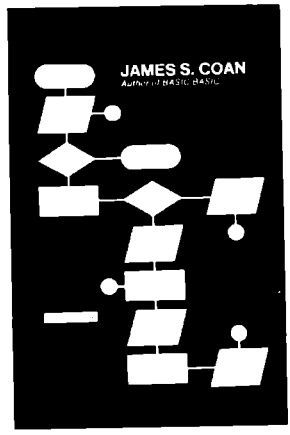
Musical Applications of Microprocessors von Hal Chamberlin.
Hal Chamberlin kann als Papst für Computermusik bezeichnet werden. In diesen 660seitigen Buch hat er sein Wissen und seine Erfahrungen der Öffentlichkeit zugänglich gemacht. Grundlagen, Einführung, Schaltungsbeisp., Programmbeisp., digitale Tonerzeugung, digitale Filter u. v. a.
Best.-Nr.: 265 **79,00 DM**



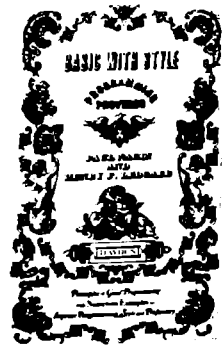
Advanced BASIC, Applications a. Problems
Kann als Fortsetzung von Nr. 255 angesehen werden. Tiefgehende Informationen über Files, Strings, Geometrie, Reihen u. Folgen, Matrizen, Statistik u. Simulationen.
Best.-Nr. 266 **39,00 DM**



How to Profit from your Personal Computer
Dieses Buch ist für den Geschäftsmann und Computer-Hobbyisten geschrieben. Es zeigt Ihnen wie Sie den Computer für sich arbeiten lassen können und dabei noch Geld verdienen. (Viele Tips und Programmbeispiele in BASIC.)
Best.-Nr. 267 **39,00 DM**



Basic FORTRAN
Vom Autor des Buches "BASIC BASIC". Eine ausgezeichnete Einführung in die Programmiersprache FORTRAN. Schritt für Schritt mit vielen Beispielen (über 80) lernen Sie diese leistungsfähige Sprache.
Best.-Nr. 271 (engl.) **45,00 DM**

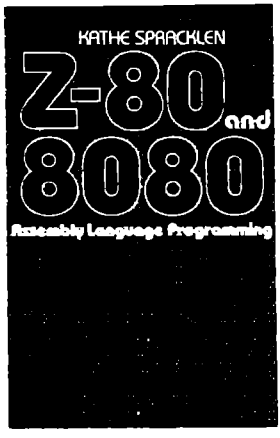


BASIC with Style
COBOL with Style
PASCAL with Style
Diese Buchserie ist eine Art "Hohe-Schule" für den ernsthaften Programmierer. Viele Tips Beispiele, nützliche Unter- und Hilfsprogramme für die Programmerstellung am Arbeitsplatz oder in der Freizeit. Top-Down Programing, Program Standard Odds and Ends sind die Hauptthemen in diesen drei sehr wertvollen Büchern. Jedes Buch ca. 145 – 200 Seiten.

PASCAL with Style
Best.-Nr. 268 **39,00 DM**

COBOL with Style
Best.-Nr. 269 **39,00 DM**

BASIC with Style
Best.-Nr. 270 **39,00 DM**

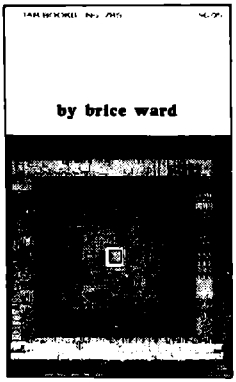


Z-80 and 8080 Assembly Language Programming
Ein Einführungsbuch in die Maschinensprache für 8080 und Z-80 Systeme. Es bringt Ihnen fast alles, was Sie zur optimalen Nutzung Ihres Computers brauchen. Ideal für Anfänger.
Best.-Nr. 272 (engl.) **39,00 DM**

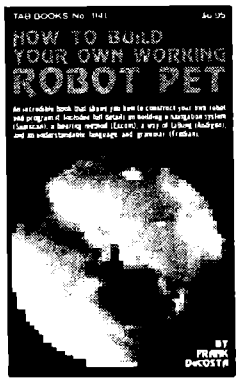
Beat the ODDS
Microcomputer Simulationen von Casino-Spielen, Trente-et-quarante, Roulette, Blackjack, Craps, Chemin-de-fer. Eine extrem nützliche Programmierhilfe für realistische Simulationen der wichtigsten Casino Spiele.
Best. Nr. 273 **39,00 DM**

Endlich ein Buch über den 1802 Microprozessor
Den leistungsfähigen 1802 Microcomputer in CMOS Technologie, der heute vorwiegend in der Industrie und KFZ-Technik eingesetzt wird und in riesigen Stückzahlen produziert wird, wurde bis jetzt wenig allgemeines Interesse entgegengebracht. Ganz zu unrecht! Den dieser Prozessor ist für das obengenannte Einsatzgebiet hervorragend geeignet. Das Buch eignet sich für den Einsteiger genau so wie für den Experten. Einführung, Grundl. der Assembler Programmierung, Befehlssatz mit Erläuterungen. Ein kompletter 1802 Assembler als Listing, Übungsbeispiele m. Antworten.
Best.-Nr. 264 **19,80 DM**

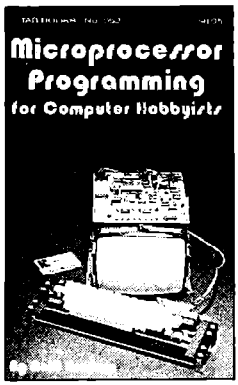
TAB-Books



Microprocessor Programming for the Computer Hobbyist
Dieses buch ist speziell für den Computer-Hobbyisten geschrieben, der sich bereits mit weiterführenden Programmiertechniken und Datenstrukturen beschäftigen möchte. Inhalt: Höhere Programmiersprachen (PL/M,PL/1), arithmetische Funktionen, Suchprogramme werden im Zusammenhang mit dem Schachspielproblem behandelt. Über 380 Seiten in englischer Sprache
Best.-Nr. 952 **DM 39,--**

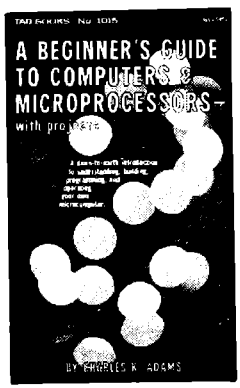


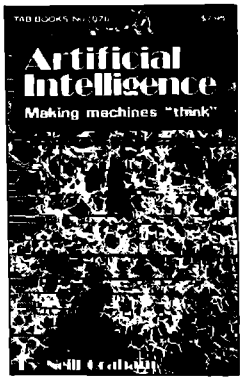
Microprocessor/Microprogramming Handbook, Brice Ward
Ein praktisches Handbuch für jeden, der sich mit Mikroprozessoren beschäftigen möchte. Auf über 290 Seiten finden Sie Grundlagen, alles über Programmierung und Anwendungsbeispiele der interessantesten integrierten Schaltungen. Inhalt: Einführung in die Mikroprozessortechnik. Der Mikroprozessor von innen. MCS4, MCS40, MCS80, Speichersysteme, Mikroprogrammierung in Maschinen- und Assemblersprache.
Best.-Nr. 785 **DM 35.--**



How to build your own working ROBOT PET
An incredible book that shows you how to construct your own robot and program it. Includes full details on building a navigation system (Soniscan), a hearing method (Excom), a way of talking (Audigen) and an understandable language and grammer (Fredian).
Eine wertvolle Hilfe für alle, die sich einen Roboter bauen wollen. (8085) CPU). Viele wertvolle Schaltungen, die Sie im Zusammenhang mit TRS-80 oder 8085-Computern verwenden können. 238 Seiten.
Best.-Nr. 1141 **DM 29,80**

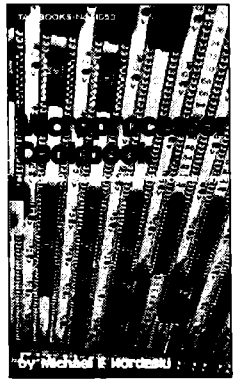
Beginner's Guide to Computer Programming, Brice Ward
Eine ideale Einführung in die Programmierung von Computern (Microprozessoren). Das Buch beginnt mit der Entwicklung einer einfachen Programmiersprache für den eigenen Bereich und zum Selbststudium, sodann wird auf andere Sprachen übergegangen. Inhalt: Grundlegende Programmkonzepte, I/O-Schaltungen, Flußdiagramme, Programmtest, Schleifen, Indexregister, versch. Programmiersprachen, Compiler, Cobol u. v. a., 480 Seiten, 364 Bilder , in englischer Sprache.
Best.-Nr. 1015 **DM 29,80**

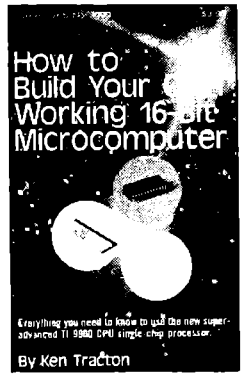




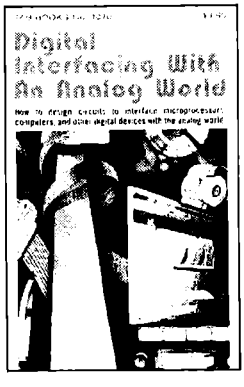
Artificial Intelligence, Neil Graham
Hier ist endlich ein Buch speziell über künstliche Intelligenz. Der Autor benutzt Computerspiele und Robotertechnik für die illustrative Behandlung dieses brandheißen Themas. Was ist der Unterschied zwischen einer Lösung einer komplexen Programmieraufgabe und der Programmerstellung für intelligente Computerentscheidungen.
Best.-Nr. 1076 **DM 29,80**

Microprocessor Cookbook, von Michael Hordeski
A chip-by-chip-comparison of todays modern microprocessors.
Die wichtigsten Prozessortypen werden genau beschrieben (Schaltung, Blockdiagramm, Befehlslisten, Programmbeispiele) und verglichen. 8080, 6800, F8, Z80, TMS 9900, SC/MP, Bit Slices, R6500, 264 S.
Best.-Nr. 1053 **DM 24,80**



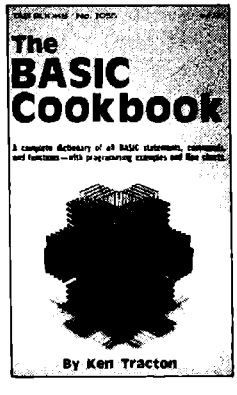


How to Build your Own Working 16-Bit-Microcomputer von Ken Tracton
Alles, was Sie über den neuen Supermicroprozessor TI9900 wissen müssen. Beschreibung des Schaltkreises, Peripherie-Bausteine für TMS9900. Tricks, Systemaufbau etc. Sehr interessant, da der TMS9900 das Herz des neuen Texas Instruments Personal Computers ist.
Best.-Nr. 1099 **DM 14,80**

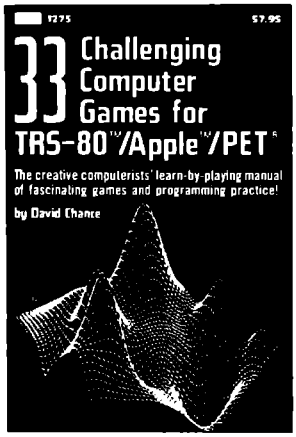


Digital Interfacing with an Analog World, Joseph J. Carr
Wie entwirft und baut man Interfaceschaltungen für Microcomputer? Der Schwerpunkt liegt bei der Verbindung mit der analogen Welt. Viele Tabellen, Schaltungen, Berechnungen und Hinweise. Über 400 Seiten.
Best.-Nr. 1070 **DM 39,--**

The BASIC Cookbook, von Ken Tracton
Ein komplettes Dictionary mit BASIC-Befehlen. Alle Befehle sind in alphabetischer Reihenfolge geordnet und jeder Befehl genau beschrieben. Zu jedem Befehl ist ein Demonstrationsbeispiel beigefügt.
Dieses Buch braucht jeder BASIC-Programmierer!
Best.-Nr. 1055 **DM 24,80**

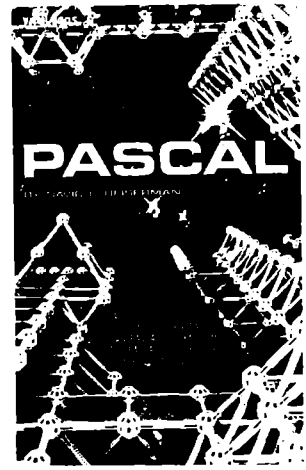


25 Games in BASIC (Listings). Die Programme laufen mit kleinen Änderungen auf den meisten Microcomputern. Inhalt: Number, Letter, Stars, Trap, Bagels, Mugwump, Hurkle, Snark, Reverse, Button Chomp, Taxman u. v. a.
Best.-Nr. 8057 **DM 9,80**



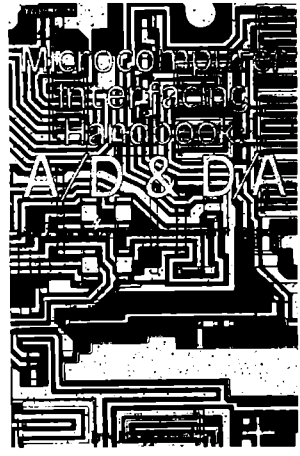
33 Challenging Computer Games for TRS-80, APPLE, PET
Das Handbuch für den Computerist, der durch Spielen lernen will. Über faszinierende Spiele zur Programmierpraxis. Ein Buch f. jeden Personal Computer-Bes.
Best.-Nr.: 1275 **29,80 DM**

Handbook of Microprocessor Applications (6800, 6802). Wie werden Microcomputer in der Praxis eingesetzt. Interfacetechniken u. Anwendung v. Maschinensprache, Schaltungsbeispiele mit Software. Beschäftigt sich in erster Linie mit 6800, 6802 und den zugehörigen Peripheriebausteinen.
Best.-Nr.: 1203 **29,80 DM**

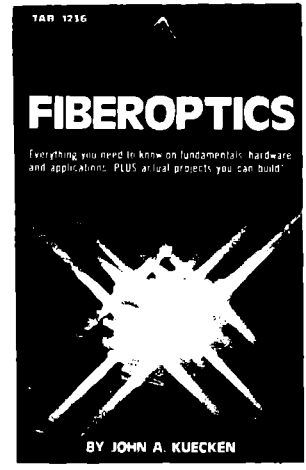


PASCAL
Eine Programmieranleitung f. d. PASCAL-Freund. Wie verwendet man TINY PASCAL und Supersoft-PASCAL. Passt ideal zu diesen Paketen und TRS-80. Viele Programmierbeispiele. (350 Seiten)
Best.-Nr. 1205 **35,00 DM**

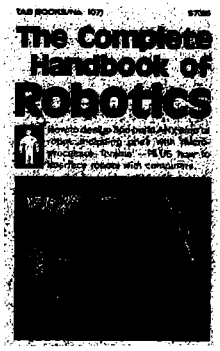
The FORTRAN Cookbook
Heute noch verwendenl eine Einführung in die FORTRAN-Programmierspr., Grundlagen, Beispiele, Übungen und Beispiele aus dem Microcomputerbereich. (Mathem. Probleme bei Computerspielen u. v. a.)
Best.-Nr.: 1187 **29,80 DM**



Microcomputer Interfacing Handbook: A/D- und D/A-Wandler
Ein Hand- u. Nachschlagebuch f. jeden der Analog/Digital od. Digital/Analog-Wandler an seinen Microcomputer anschließen will. Unentbehrlich f. die Anwendung des Microcomputers im industriellen Einsatz.
Best.-Nr.: 1271 **35,00 DM**

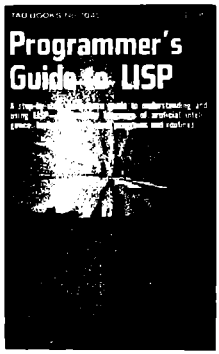


FIBEROPTICS
Alles was Sie über d. Grundlagen, Hardware und Applikationen dieser faszinierenden neuen Technik wissen sollten. Mit Projekten, die Sie selbst bauen können. (Übertragungsraten bis zu 1.354 x 10^9 Bildpunkte pro Sekunde, das bringt die Fiberoptic!)
Best.-Nr.: 1236 **29,80 DM**

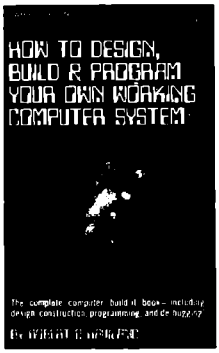


The Complete Handbook of Robotics
E. Safford, jr.
Wie entwickle und baue ich einen Roboter. Es werden auch microcomputergesteuerte Roboter beschrieben, und wie man Roboter mit Computern zusammenschaltet. Terminologie, Entwicklungsstand, Sensoren, Grundlagen , das Roboter "Gehirn", Servomechanismus, Tips, Interfacing u. v. a.
Best.-Nr. 1071 **DM 29,80**

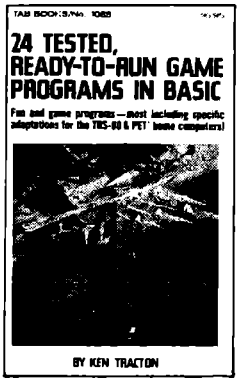
1001 Things to do with your Personal Computer, Mark Sawusch
Computeranwendungen für jedermann, Geschäft und Finanzprogramme, mathematische Applikationen, technisch/wissenschaftliche Anwendung von Personal Computern, Anwendungen in Erziehung und Ausbildung, Hobby Computer, Spiele, Control- und periphere Applikationen, künstliche Intelligenz. Viele Programme in BASIC, viele Tips. 330 Seiten
Best.-Nr. 1160 **DM 29,80**



Programmer's Guide to LISP, K.Tracton
Eine Einführung in die Computersprache LISP. LISP ist eine spezielle Interpreterversion, die sich gut für die Programmierung im Bereich künstliche Intelligenz eignet. Das Buch zeigt Ihnen Wege und Lösungen. Viele Programmbeispiele. 210 Seiten.
Best.-Nr. 1045 **DM 24,80**

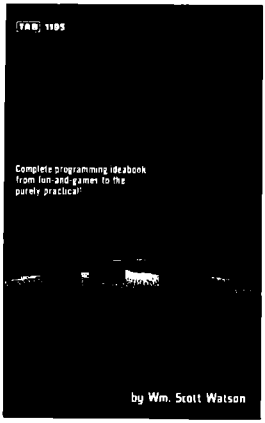


How to Design, Build and Program your own Working Computer System
R. Haviland
Dieses Buch zeigt Ihnen Schritt für Schritt, wie Sie einen eigenen Computer aufbauen können. Es lehnt sich dabei an den erfolgreichen SC/MP von National an. Ein Buch, das jeder SC/MP-Besitzer haben muß.
Best.-Nr. 1111 **DM 29,80**

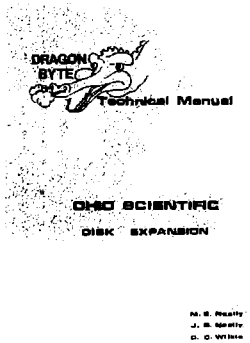


24 Tested Ready to RUN Game Programs in BASIC, Ken Tracton
Spaß- und Spielprogramme in BASIC. Viele Programme enthalten spezielle Anpassungshinweise an die Homecomputertypen TRS-80 und PET. Viele Graphik- und Zeichenprogramme. Ideal für jeden BASIC-Computer-Besitzer.
Best.-Nr. 1085 **DM 24,80**

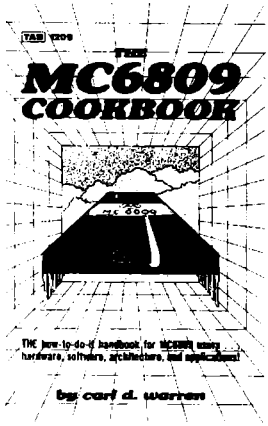
34 MORE Tested Ready-To-Run Game Programs in BASIC
34 Spielprogramme für Video Genie und TRS-80, aber auch für andere BASIC Personalcomputer. Ein Buch, welches Ihnen viel Freude mit Ihrem Computer bereiten wird.
Best.-Nr. 1228 **35,00 DM**



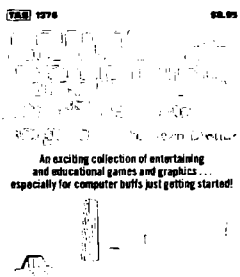
67 Ready to Run Programs in BASIC
Ein Buch mit vielen Programmbeispielen für alle die einen BASIC-Computer haben (ab 4K RAM). Viele Tips, Anregungen und Hilfen bringen auch dem Erstanwender großen Nutzen.
Best.-Nr. 1195 **29,80 DM**



Dragon Byte Disk Expansion Book
Ein Büchlein für den, der seinen Ohio Scientific Computer zum Diskettensystem selbst ausbauen will. Sehr gute technische Informationen, die Sie sonst nur schwer finden.
Best.-Nr. 311 **29,80 DM**

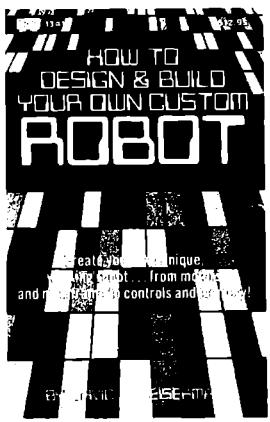


The MC 6809 Cookbook
Endlich ein ausführliches Handbuch für den extrem Leistungsfähigen 6809 Prozessor, Hardware, Software, Architektur. Wie macht mans? Von Carl Warren.
Best.-Nr. 1209 **29,80 DM**



Computer Graphics with 29 ready-to-run Programs
Eine Einführung in die Microcomputergrafik für Anfänger. Eine wahre Fundgrube für denjenigen der gute Programmbeispiele mit Grafik sucht. (Ideal für Video Genie und TRS-80 Level II BASIC.)
Best.-Nr. 1276 **39,00 DM**

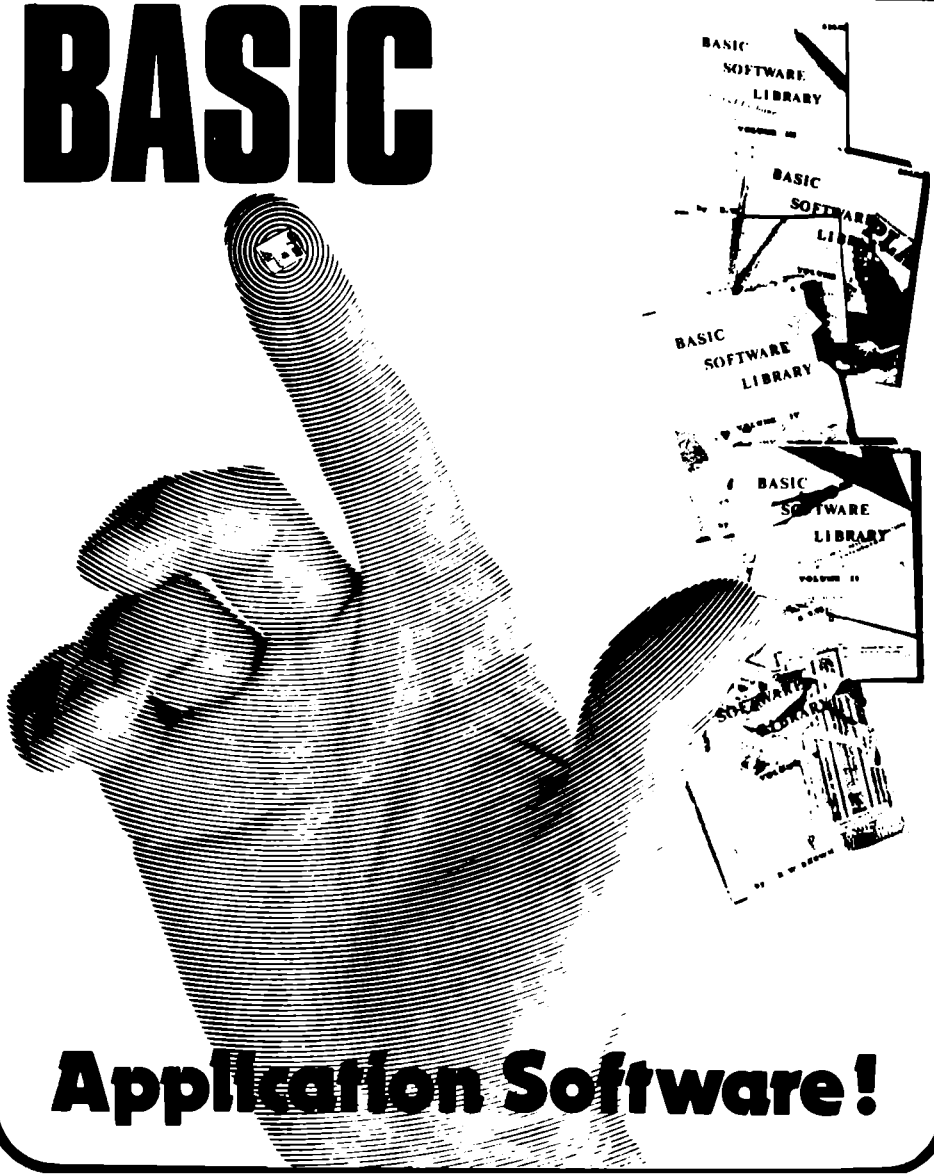
How to build your own working Microcomputer
Dieses Buch zeigt Ihnen wie Sie sich selbst einen einfachen Microcomputer bauen können. Programmspiele, Funktionsdiagramme, Schaltbilder. Auch für Anfänger geeignet.
Best.-Nr. 1200 **49,00 DM**



How to Design and Build your own Custom Robot
Bauen Sie Ihren eigenen Roboter, der Ihre Arbeiten erledigen kann. Einführung, elektrische und mechanische Grundlagen, Microcomputer-Hardware (8085 und 780) u. v. a.
Best.-Nr. 1341 **59,00 DM**

PROGRAMM BIBLIOTHEK

BASIC



Application Software!

INHALTSVERZEICHNIS DER BASIC PROGRAMMBIBLIOTHEK VOL I - VOL V

VOLUME ONE

Part 1 - Business + Personal Bookkeeping Programs

Name	Description
Bond	Computers price and interest for bond purchases.
Building	Analyzes the cost of building design proposals.
Compound	Computes effective compound interest rates.
Cyclic	Determines seasonal coefficients for two cycles.
Decision 1	Makes a Lease/Buy decision for you.
Decision 2	Makes a decision on whether to buy a component or make it.
Depreciation	Calculates depreciation by 4 different methods.
Efficient	Cal. the most efficient assignment of resources and / or personnel.
Flow	Predicts your yearly cash flow.
Installment	Performs monthly installment accounting.
Interest	Computes interest accruals, monthly.
Investments	Computes annual rates of return on investments.
Mortgage	Makes a comparison of mortgage terms.
Optimize	Optimizes the layout for a plant, shop, office, etc.
Order	Determines your economic order quantity for inventory items.
Pert Tree	Performs an analysis of a pert network.
Rate	Computes true annual interest rates.
Return 1	Computes lessor's rate of return for uncertain assets.
Return 2	Computes a lessor's rate of return after taxes.
Schedule 1	Schedules N jobs in a shop with M machines.

Part 2 - Games + Pictures

Name	Description
Animals Four	Teach the computer all about animals.
Astronaut	Land your spaceship on another planet.
Bagel	Advanced number game, numbers may be algebraic, few clues.
Bio Cycle	Calculate your Bio-Life Cycle and plan your days.
Cannons	An advanced war game with big guns.
Checkers	Plays a regulation game of checkers.
Craps	A dice game with hard way odds.
Dogfight	Air fight w/missiles; between a phantom and a mig.
Golf	Plays any number of holes; inc. obstacle course.
Judy	Have a rap session with Judy via your computer.
Line Up	Simple number game, all you have to do is unscramble them.
Pony	Authentic horse race, any number of players.
Roulette	Gamblers delight, plays Las Vegas rules.
Sky Diver	Sky dive on another planet
Tank	A war game between two tanks.
Teach Me	Teach the computer to learn new things.
A Newman	He's absolutely MAD! MAD! MAD!
J. F. K.	Our 35th. president.
Linus	Loveable "Peanuts" character, w/blanket.
Ms. Santa	A modern miss to put a twinkle in your eye.
Nixon	Former "United States" president.
Noel Noel	Christmas or anytime this is a beautiful creation.
Nude	A true work of art for anyone's gallery.
Peace	A message for all seasons.

Name	Description
Policeman	True and blue, he's the law.
Santa's Sleigh	In banner form, perfect for decorating the mantle.
Snoopy	That paragon of Dogdom even plays football.
Virgin	A picture you can read as well as see.

VOLUME TWO

Part 3 - Math + Engineering Programs

Name	Description
Beam	Evaluates and selects steel beam sizes.
Conv.	Calculates convolutions.
Filter	Calculates low pass filter components.
Fit	Performs interpolations by spline fits.
Integration 1	Uses Gaussion Quadrature to do integration.
Integration 2	Integrates a function by spline fits.
Intensity	Calc. and plots RF or Acoustic intensities.
Lola	Calc. Long. and Lat. from interstellar fix or distance.
Macro	Simulates a language compiler.
Max. Min.	Calc. the max. + min. values of funct. over a special interval.
Navaid	Calc. position from altitude and azimuth of celestial bodies.
Optical	Calculates Blackbody energies, w/filter look-up tables.
Planet	Calculates Sun and Moon positions, hourly.
PSD	Calculates Power Spectral Densities and FFT's.
Rand 1	Generates random numbers between 0 and 1.
Rand 2	Generates random integers between (X) and (Y).
Solve	Solves polynominals by "Bairstows Methods".
Sphere Trian	Solves any spherical triangle.
Stars	Locates 50 stars (celestial).
Track	Calc. course and distance and incremental vectors.
Triangle	Solves for all parts of any triangle.
Variable	Finds all variables in BASIC programs.
Vector	Calc. final position; given start and motion vectors

Part 4 - Plotting + Statistics Programs

Binomial	Calculates binomial probability distributions.
Chi-Sq.	Applies the Chi-Square test to samples.
Coeff	Calc. coefficients of fourier series to apprx. a function.
Confidence 1	Calculates confidence limits on linear regressions.
Confidence 2	Calculates confidence limits for a sample mean.
Correlations	Performs auto and cross correlations with plots.
Curve	Fits 6 different curves by the least squares method.
Differences	Calculates difference of means in non-equal variances.
Dual Plot	Plots two functions on the same sheet.
Exp-Distri	Calculates exponential distributions for a sample.
Least Squares	Performs least squares fit by linear, exp., or power function.
Paired	Compares 2 groups of data using the rank test.
Plot	Plots 6 equations on the same sheet.
Plotpts	Plots data points on standard teletypes.
Polynomial Fit	Performs least squares polynomial fit.
Regression	Performs multiple linear fit with or without transformations.
Stat 1	Finds the mean, variance and standard deviations.

Name	Description
Stat 2	Computes various stat. measures for a variable.
T-Distribution	Calculates normal and T-distributions.
Unpaired	Compares 2 groups of unpaired data.
Variance 1	Performs one way analysis of variances.
Variance 2	Analyzes a variance table of one way random design.
XY	Plots functions of X and Y.

APPENDIX A - BASIC STATEMENT DEFINITIONS

VOLUME THREE

Part 5 - Advanced Business Programs

Billing	Performs posting and billing of accounts.
Inventory	Maintains data for inventory records.
Payroll	Computes payrolls with full set of deductions.
Risk	Performs a risk analysis on capital investments.
Schedule 2	Performs the most effi. scheduling of men or resources to loca.
Shipping	Solves the problem of scheduling and assignments.
Stocks	Computes the value of stocks.
Switch	Calculates the effects of a bond switch.

VOLUME FOUR

General Purpose Programs

Bingo	An age old favorite. "B9, C23, D4, E13, F21, BINGO!
Bonds	Computes the yields for a bond for different periods.
Bull	If you ever dreamed of being a Matador, here's your chance.
Enterprise	Take charge of the Enterprise while Capt. Kirk is on leave.
Football	Authentic NFL version of this well known sport.
Funds 1	Calculates long-term predictions of funds.
Funds 2	Plots the results of Funds 1.
Go-Moku	Ancient Chinese game of chance.
Jack	Plays Blackjack, Las Vegas style.
Life	Life is truly a battle for survival, a real challenger!
Loans	Calculates annuities, loans and mortgages.
Mazes	Generates unique maze puzzles for you to solve.
Poker	Five card draw - for up to 5 players.
Popul	Performs population projections for defined areas.
Profits	Determines the profitability of a firms various depts.
Qublic	3-Dimensional Tic-Tac-Toe.
Rates	Calc. the effective annual interest rate for stated interest.
Retire	Calculates your Civil Service Retirement benefits.
Savings	Computes savings plan profiles.
SBA	Calculates repayment schedules for SBA loans.
Tic-Tac-Toe	An all time favorite for young and old alike.

VOLUME FIVE

Experimenter's Programs

Andy Cap	Draws this famous cartoon character.
Baseball	Plays a full 9 innings of baseball.
Compare	Compares two groups of data.
Confid 10	Determines the confidence limits for a normal population.
Descrip	Provides a description of uni-variant data.
Differ	Computes the diff. of the means for data of equal variance.
Engine	Calculates the otto cycle of engines.
Fourier	This program evaluates fourier series.
Horse	Draws a picture of a horse.
Integers	Computes integers as the sum of other integers.

Name	Description
Logic	Determines conclusions from logic statements.
Playboy	Draws the playboy symbol.
Primes	Factors numbers into their primes.
Probal	Calc. Chi-Sq. and probabilities from 2X2 data sets.
Quadrac	Solves quadratic equations
Red Baron	Draws a picture of the infamous Red Baron.
Regression 2	Calculates linear regressions.
Road Runner	"Beep! Beep!" Draws a picture of the Road Runner.
Roulette	Computerized "Wheel of Fortune", plays roulette.
Santa	Old Saint Nick appears as jolly as ever.
Stat 10	Calculates quantities for two groups of paired data.
Stat 11	Computes sample statistics.
Steel	Calculates steel beam capacities.
Top	Computes cost for surfacing a road or driveway, etc.
Vary	Performs an analysis of a vari. table; one-way random design.
Xmas	Generates a "SINGING" Christmas card.

APPENDIX B - STATEMENT CONVERSION ALGORITHMS

VOLUME SIX

A Complete Business System

Ledger	Maintains ALLCompany accounts and generates ALL financial reports. Includes routines for: Pyrl, Inv. Depr. A/R, A/P, Balance Sheets and Profit + Loss statements, etc.
ACBS rev: 80	Users Manual - A Proprietary Package

VOLUME SEVEN

Professional Programs

Chess	Designed to challenge the average player, fairly comprehensive. Great fun for all, offers a unique opportunity for beginners in need of an opponent.
Medbil	For Doctors and Dentists alike, a complete patient billing system which also permits the maintaining of a patient history record.
Wdproc	Wordprocessing for lawyers, publishers, writers etc. Write, store and change from rough draft to final copy in a variety of formats.

VOLUME SEVEN

Professional Programs

Utility

Licensing Agreement

BESTELLSCHEIN

Heute noch an Ihren Buch- oder Fachhändler absenden! Bitte senden Sie mir folgende Bücher:

..... Stck. Vol. I , Best.Nr. 80/50, DM 99,--
..... Stck. Vol. II , Best.Nr. 80/51, DM 99,--
..... Stck. Vol. III, Best.Nr. 80/52, DM 149,--
..... Stck. Vol. IV, Best.Nr. 80/53, DM 39,--
..... Stck. Vol. V , Best.Nr. 80/54, DM 39,--
..... Stck. Vol. VI, Best.Nr. 80/48, DM 199,--
..... Stck. Vol. VII, Best.Nr. 80/49, DM 159,--
..... Stck. Programm Bibliothek BASIC, (Volume I bis Vol. V) Best.Nr. 80/21, DM 425,--

Hofacker Verlag

GRUNDAUSSTATTUNG

Die Grundausstattung der BASIC Programmbibliothek besteht aus fünf Büchern mit insgesamt 1100 Seiten Programmlistings und genauen Beschreibungen. (Format DIN A 4)
Sie finden in diesem Werk 149 verschiedene BASIC Programme aus folgenden Bereichen:

- Buchhaltung
- Computerspiele
- Bildprogramme
- Mathematik und Ingenieurwissenschaften
- Statistik, technische Mechanik etc.
- Commerzielle Programme, Rechnungswesen, Fakturierung
- Inventurprogramme, Lohn-und Gehaltsbuchhaltung
- Experimentierprogramme u.v.a.

Jedes Programm ist genau beschrieben.
Die fünf Bände bilden eine komplette "Do It Yourself" Programmbibliothek, die sehr einfach zu handhaben ist. Es werden jedoch Grundkenntnisse in der Programmierung vorausgesetzt.

PREISSENSATION

Bei 149 Programmen in dieser umfangreichen Bibliothek bezahlen Sie nur
DM 2.85 pro BASIC Programm - Das ist einmalig!

BESCHREIBUNG DER PROGRAMMBIBLIOTHEK

Diese Bibliothek ist die umfangreichste und universalste ihrer Art auf dem Weltmarkt. Die Besonderheit dieser fünf Werke liegt darin, daß dem Anwender die Möglichkeit gegeben wird, in erster Linie sinnvolle und produktive Aufgaben wie Buchhaltung, Statistik, mathematische Probleme etc., zu lösen. Computerspiele sind jedoch auch enthalten.

Alle Programme wurden mehrmals auf verschiedenen Systemen ausgeführt und getestet. Jedes Programm ist mit einer genauen Beschreibung und einer Aufstellung der infrage kommenden Anwender versehen. Befehle und mögliche Grenzen der Programme sind auch aufgezeigt, wenn die Programme auf unterschiedlichen Systemen ausgeführt werden sollen. Auch der benötigte Speicherbereich ist für jedes Programm genau angegeben.
Jedes Programm ist in seiner vollständigen Größe aufgeführt und nicht reduziert. Es ist in sich selbst vollständig und kann im Ablauf genau verfolgt werden. Es gibt Aufschluß über alle wichtigen Daten. Bei den meisten Programmen folgt unmittelbar nach dem Programmlisting ein Probelauf. Die meisten Programme sind in kompatiblem BASIC geschrieben, welche auf den meisten 4 K BASIC-Versionen ausgeführt werden können.

Best.Nr. 80/21, 5 Bücher mit 1100 Seiten DIN A 4 (149 Programme) nur DM 425,--

Die Bände sind auch einzeln erhältlich:

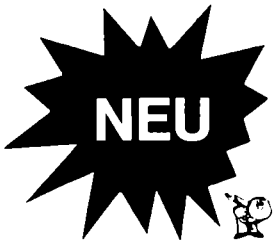
BASIC Volume I
Programme über Buchhaltung, Computerspiele und Bilder (Computerbilder per Programm)
Best.Nr. 80/50 DM 99,--

BASIC Volume II
Programme über mathematische Probleme, technische Programme, Zeichnen, Plotting und Statistik. Grundlegende Statements.
Best.Nr. 80/51 DM 99,--

BASIC Volume III
Erweiterte Geschäftsprogramme für den kommerziellen Bereich. Fakturierung, Inventuren, Gehaltsbuchhaltung.
Best.Nr. 80/52 DM 149,--

BASIC Volume IV
BASIC Programme aus allen Bereichen. Universelle Anwendung.
Best.Nr. 80/53 DM 39,--

BASIC Volume V
Experimentierprogramme.
Best.Nr. 80/54 DM 39,--



BASIC Volume VI
Ein komplettes, interaktives Programmpaket für Floppy Disk - Einsatz im kommerziellen Bereich. Gehaltsabrechnung, Inventur, Gewinn und Verlustrechnung u. v. a.
Best.Nr. 80/48 DM 199,--

BASIC Volume VII
Das Schachprogramm, auf das Sie schon lange gewartet haben. Spielt zwei Farben. Leichter bis mittlerer Schwierigkeitsgrad. Kann einfach erweitert werden. Zugzeit ab 3 Minuten.
Best.Nr.
Medical Billing package (Patientengeschichte und Kostenübersicht für Ärzte)
Disk interaktives Textverarbeitungssystem.
Best.Nr. 80/49 DM 159,--

Genaue Beschreibung des BASIC Volume VII

Schachprogramm (Chess)
Diese Version eines Schachprogrammes in BASIC ist für die meisten 12 K BASIC-Versionen (mit oder ohne kleinen Änderungen) geeignet. Weiterhin sollten noch ca. 12 K im RAM-Bereich zur Verfügung stehen. Das Programm spielt Schach im Schwierigkeitsgrad für Anfänger, kann aber durch entsprechende Änderungen auch schwieriger gestaltet werden. Züge werden im Bereich von 3 Minuten oder mehr durchgeführt. Der erste Zug benötigt ca. 3 Minuten. Die Zugzeit steigt dann entsprechend dem Spielfortschritt entsprechend an. Das Programm kann wahlweise schwarz oder weiß spielen. Das Spielfeld wird auf dem Bildschirm angezeigt oder kann auf einem Drucker ausgegeben werden.

BASIC-Programm für die Arztpraxis (Medbil)
Dieses Programm wurde zur Erleichterung der so teuren und mühsamen Alltagsarbeit der Ärzte entwickelt. Dieses Programm ermöglicht die Speicherung und Durchsicht jeder Patientengeschichte auf dem Bildschirm, die vorher in einer Datenbank gespeichert wurde. Weiterhin besteht eine Möglichkeit der schnellen Überprüfung von aufgelaufenen Behandlungskosten pro Patient. Auf einer 250 K Byte Diskette können etwa 110 Patienten mit all ihren Daten aufgenommen werden. Mit einer Minifloppy (80 K Byte) reduziert sich die mögliche Patientenzahl auf ca. 25 pro Diskette.

Textverarbeitungsprogramm (Wrdpro)
Dieses Programm ermöglicht dem Besitzer von Microcomputern Texte zusammenzustellen und in beliebigen Formaten wieder auszudrucken. Das Programm läuft auf den meisten extended BASIC-Versionen mit mindestens 15 K freiem RAM-Bereich.

W. Hofacker Verlag

ELCOMP

POSTKARTE

Absender
Bitte deutlich ausfüllen

Vorname/Name

Beruf

Straße/Nr.

Plz Ort

ELCOMP

MIKROCOMPUTER BOOK STORE

Tegernseerstr. 18

D-8150 Holzkirchen /Obb.

ABSENDER:

.....

Name, Vorname

.....

Straße

( ) .....

PLZ Ort

.....

Telefon

Das Abonnement wird automatisch verlängert
(Kündigung 8 Wochen z. Abonnement-Ablauf)

ELCOMP

Ing. W. Hofacker GmbH
Tegernseer Straße 18
D-8150 Holzkirchen

Hiermit gebe ich folgende Bestellung unter Anerkennung Ihrer Lieferungs- und Zahlungsbedingungen auf.

ELCOMP

Menge	Artikelbezeichnung	Einzelpreis	Menge	Artikelbezeichnung	Einzelpreis

Unterschrift (für Jugendliche unter 18 J. der Erziehungsberechtigte) Datum

BESTELLKARTE für ein ELCOMP-Abonnement

Auslands-Abonnement (Europa) + DM 10.-- Versandkosten

- ○ Bitte senden Sie mir ab sofort alle künftigen Ausgaben der ELCOMP. Preis DM 69,– pro Jahr incl. Versand und Verpackung. Lieferung ab Ausgabe.....
- ○ Bitte senden Sie mir alle Hefte von Sept. 1978 – Okt. 1979 (außer die Hefte Nr. 2 und 4 1979) zum Sonderpreis von 37,– DM
- ○ Aus dem Jahrgang 1980 die Hefte 3, 4, 5, 6, und 10 zum Preis von 20,– DM
- ○ Ich bezahle im voraus für.....Jahr(e) DM....., Pschk. München 15994807, Dresdner Bank München Kto. 5390 370, BLZ 700 800 00.
- ○ Ich bitte um Lieferung auf Rechnung.
- ○ Ich bitte um Abbuchung von meinem Konto (Giro od. Postscheckkonto)
  Kto.Nr..... BLZ.....
  Geldinstitut:..... Ort:.....

Name/Vorname

Straße / Hs-Nr.

PLZ Ort

Datum..... Unterschrift.....

****
* phs /SLS EDV - Beratung *
* Davenstedter Str. 8 3000 Hannover 91 Tel: 0511-45 38 17 *
****

Wir liefern Compiler für cbm :

PASCAL
- Vollcompiler nach ISO-Normierung
- weitgehend UCSD-kompatibel
- voller Jensen/Wirth Standard
- 11-stellige Arithmetik
- PEEK/POKE
- Viertelpunktgraphik über PLOT(x,y,boolean)
- virtuelles RAM bis 1 Megabyte mit cbm 8050
- Dateidirektzugriff
- jegliche Peripherie wie gewohnt ansprechbar (IN- und OUT !)
- lauffähig mit 32 kByte RAM und Doppelfloppy
- Sonderversion mit 64 kByte für 8096 lieferbar

BASIC
- Vollcompiler mit BASIC 4.0 **und** ExBasic Level II
- automatische Programmoptimierung
- darüberhinaus erweiterte Standard-BASIC-Funktionen
- dynamische RAM-Verwaltung
- verschiebbarer Code
- Linker für Assemblerprogramme

FORTH
- vollständiges FORTH-Betriebssystem (passend zu diesem Buch)
- Diskhandling
- Macro-Assembler
- 398 Befehle bereits implementiert
- Linker für Assemblerprogramme

Bei allen Compilern handelt es sich um ausgereifte Systeme, die gerade für den kommerziellen Nutzer alle Möglichkeiten der Commodore-Geräte ohne Einschränkungen nutzbar lassen (auch fremde Hard- oder Softwareerweiterungen).
Die Compiler werden auf Diskette geliefert, deshalb bei allen Bestellungen unbedingt Rechner- und Floppytype angeben !
Unsere Preise : Jeder Compiler DM 698,-- inkl. 13 % MWSt. bei Kauf
DM 113,-- inkl. 13 % MWSt. bei 14 Tagen Miete
(nur PASCAL oder BASIC !)

---

phs/SLS Absender:.....

Davenstedterstr. 8 .....

3000 Hannover 91 .....

Betrifft: cbm-Compiler

Ich besitze einen cbm 3032/4032 KS/4032 GS/8032/8096
sowie eine Floppy cbm 3040/4040/8050
Mich interessiert:
FORTH PASCAL BASIC Sonstiges Programm

Ich möchte
- nähere Informationen
- den Compiler für 14 Tage zur Miete (DM 113,--)
- den Compiler kaufen (DM 698,--)
- Ihren Katalog (DM 15,--)

Der Betrag von DM .....
- soll per Nachnahme erhoben werden
- liegt als Verrechnungsscheck bei

Weitere interessante Bücher von Hofacker:

Best.-Nr.	Titel	Preis/DM
	Bücher in deutscher Sprache	
1	Transistor Berechnungs- u. Bauanleitungsbuch – 1	29,80
2	Transistor Berechnungs- u. Bauanleitungsbuch – 2	19,80
3	Elektronik im Auto	9,80
4	IC-Handbuch, TTL, CMOS, Linear	19,80
5	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	IC-Schaltungen, TTL, CMOS, Linear	19,80
7	Elektronik Schaltungen	9,80
8	IC-Bauanleitungs-Handbuch	19,80
9	Feldeffekttransistoren	9,80
10	Elektronik und Radio	19,80
11	IC-NF Verstärker	9,80
12	Beispiele Integrierter Schaltungen (BIS)	19,80
13	HEH, Hobby Elektronik Handbuch	9,80
14	IC-Vergleichsliste	29,80
15	Optoelektronik Handbuch	19,80
16	CMOS Teil 1, Einführung, Entwurf, Schaltbeispiele	19,80
17	CMOS Teil 2, Entwurf und Schaltbeispiele	19,80
18	CMOS Teil 3, Eintwurf und Schaltbeispiele	19,80
19	IC-Experimentier Handbuch	19,80
20	Operationsverstärker	19,80
21	Digitaltechnik Grundkurs	19,80
22	Mikroprozessoren, Eigenschaften und Aufbau	19,80
23	Elektronik Grundkurs, Kurzlehrgang Elektronik	9,80
24	Microcomputer-Technik	29,80
25	Hobby Computer Handbuch	29,80
26	Mikroprozessor, Teil 2	19,80
27	Mikrocomputer Software Handbuch	29,80
28	Lexikon + Wörterb. f. Elektr. u. Mikroprozessor LEM	29,80
29	Mikrocomputer Datenbuch	49,80
30	Aktivtraining Mikrocomputer	49,80
31	57 Programme in BASIC	39,00
32	ATARI BASIC Handbuch	29,80
33	Microcomputer Programmierbeispiele	19,80
34	TINY-BASIC Handbuch	19,80
35	Der freundliche Computer	29,80
103	Oszillographen-Handbuch	19,80
104	1000 Elektronik Schaltungen	49,00
107	Praktische Antennentechnik	19,80
108	SC/MP Mikrocomputer-Handbuch	29,80
109	6502 Microcomputer Programmierung	29,80
110	Programmierhandbuch für PET	29,80
111	Programmieren mit TRS-80	29,80
112	PASCAL-Programmier-Handbuch	29,80
113	BASIC-Programmier-Handbuch	19,80
114	Der Microcomputer im Kleinbetrieb	39,80
115	6809 Programmier Handbuch	49,00
116	Einführung 16-Bit Microcomputer	29,80
117	FORTRAN für Heimcomputer	19,80
118	Programmieren in Maschinensprache mit dem 6502	49,00
119	Programmieren in Maschinensprache (Z80)	49,00
120	Anwenderprogramme für TRS-80 u. Video Genie	29,80
121	Microsoft BASIC-Handbuch	29,80
122	BASIC für Fortgeschrittene	39,00
123	IEC-Bus Handbuch	19,80
124	Programmieren in Maschinensprache mit CBM	19,80
127	Einführung i. die Microcomputer-Progr. mit 6800	49,00
128	Programmieren mit dem CBM	29,80
129	ELCOMP-Leser Programmierhandbuch	69,00
130	Programmierbeispiele für CBM	19,80
131	COBOL für Anfänger	19,80
132	CP/M-Handbuch	19,80
133	Welches Betriebssystem brauche ich ?	19,80
139	BASIC für blutige Laien	19,80
140	ZX81 Programmierhandbuch	29,80
141	Programmierhandbuch für VC-20	29,80
	Bücher in englischer Sprache	
150	Care and Feeding of the Commodore PET	19,80
151	8k Microsoft BASIC Reference Manual	19,80
152	Expansion Handbook for 6502 and 6800	19,80
153	Microcomputer Application Notes	29,80
154	Complex Sound Generation using the SN76477	19,80
155	The First Book of 80-US (TRS-80)	19,80
156	Small Business Programs	29,80
157	The First Book of Ohio Scientific	19,80
158	The Second Book of Ohio Scientific	19,80
159	The Third Book of Ohio Scientific	19,80
160	The Fourth Book of Ohio Scientific	29,80
161	The Fifth Book of Ohio Scientific	19,80
162	ATARI Games in BASIC	19,80
163	The Peripheral Handbook	29,80
164	ATARI-BASIC Learning by Using	19,80
1050	The Most Popular Subroutines in BASIC	24,80
1053	Microprocessor Cookbook	24,80
1055	The BASIC Cookbook	24,80
1062	The A to Z Book of Computer Games	29,80
1070	Digital Interfacing with an Analog World	39,00
1071	The Complete Handbook of Robotics	29,80
1076	Artifical Intelligence	29,80
1085	24 Tested Ready to RUN Game Programs in BASIC	24,80
1088	Illustrated Dictionary of Microcomputer Terminology	35,00
1095	Programs in BASIC for Electronic Engineers	19,80
1099	How to Build Your own Working 16-Bit Microc.	14,80
1141	How to Build Your own Working ROBOT PET	29,80
1160	1001 Things to do with Y.P.C.	29,80
1169	The Giant Book of Computers	39,00
8029	Z-80 Assemblerhandbuch	29,80
8042	6500 Software Manual	19,80
8043	6500 Hardware Manual	19,80
8048	BASIC Software Vol. VI	199,00
8049	BASIC Software Vol. VII	159,00
8050	BASIC Software Vol. I	99,00
8051	BASIC Software Vol. II	99,00
8052	BASIC Software Vol. III	149,00
8053	BASIC Software Vol. IV	39,00
8054	BASIC Software Vol. V	39,00

HOFACKER

HOLZKIRCHEN **LOS ANGELES**